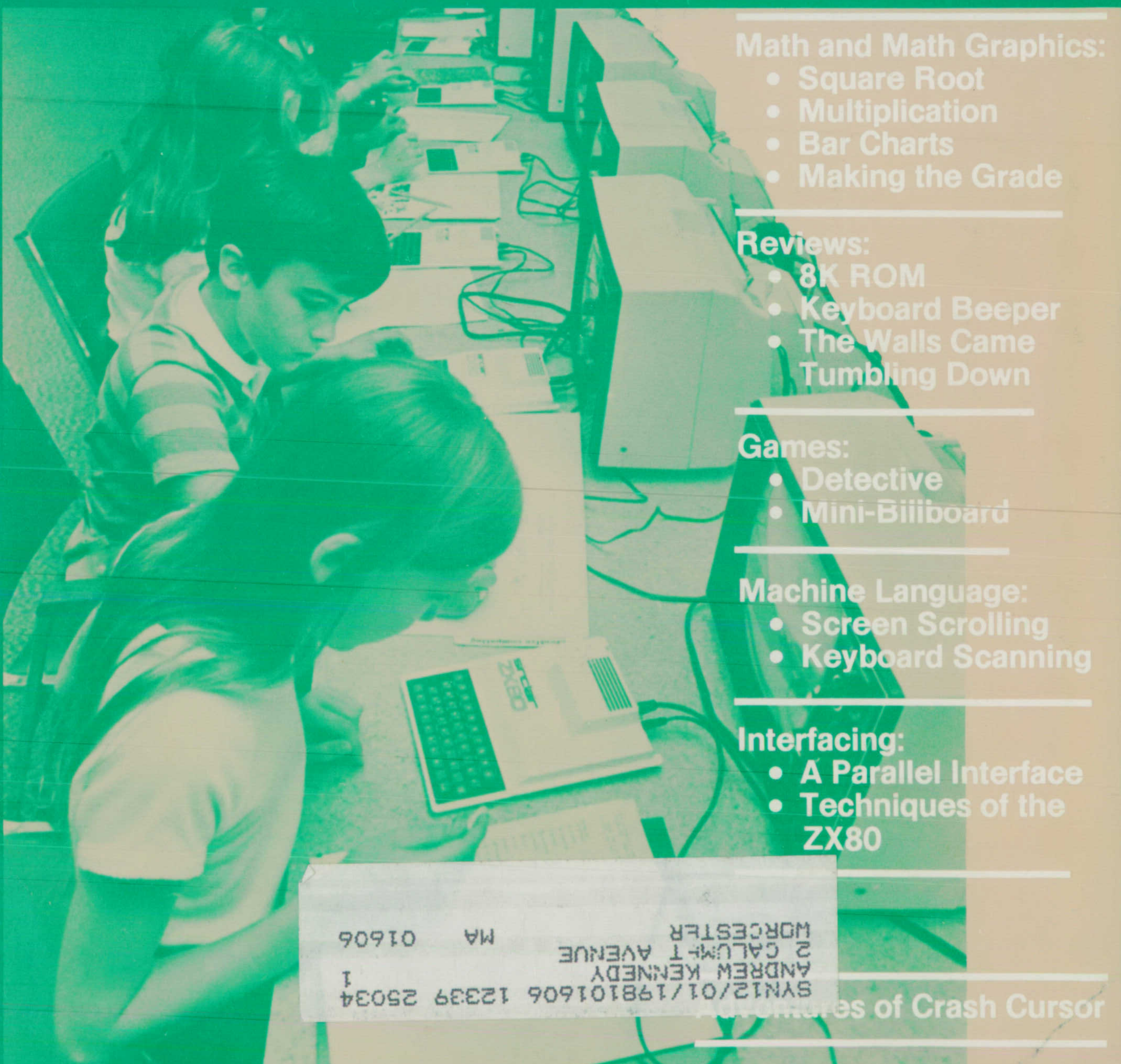
July/August 1981 Volume 1, Number 4 $2.50 (USA) £1.20 (UK)

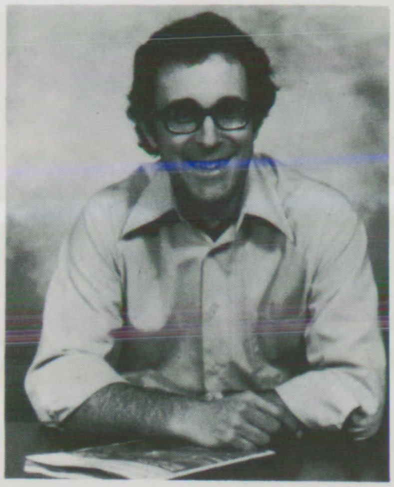
The magazine for Sinclair ZX80/1 users

SYNC



SYN12/01/19810160б 12339 25034
ANDREW KENNEDY 1
2 CALUMET AVENUE
WORCESTER MA 01606

A REMARKABLE MAGAZINE



David Ahl, Founder and Publisher of Creative Computing

creative computing

"The beat covered by Creative Computing is one of the most important, explosive and fast-changing."—Alvin Toffler

You might think the term "creative computing" is a contradiction. How can something as precise and logical as electronic computing possibly be creative? We think it can be. Consider the way computers are being used to create special effects in movies—image generation, coloring and computer-driven cameras and props. Or an electronic "sketchpad" for your home computer that adds animation, coloring and shading at your direction. How about a computer simulation of an invasion of killer bees with you trying to find a way of keeping them under control?

Beyond Our Dreams

Computers are not creative per se. But the way in which they are used can be highly creative and imaginative. Five years ago when *Creative Computing* magazine first billed itself as "The number 1 magazine of computer applications and software," we had no idea how far that idea would take us. Today, these applications are becoming so broad, so all-encompassing that the computer field will soon include virtually everything!

In light of this generality, we take "application" to mean whatever can be done with computers, *ought* to be done with computers or *might* be done with computers. That is the meat of *Creative Computing*.

Alvin Toffler, author of *Future Shock* and *The Third Wave* says, "I read *Creative Computing* not only for information about how to make the most of my own equipment but to keep an eye on how the whole field is emerging.

Creative Computing, the company as well as the magazine, is uniquely lighthearted but also seriously interested in all aspects of computing. Ours is the magazine of software, graphics, games and simulations for beginners and relaxing professionals. We try to present the new and important ideas of the field in a way that a 14-year old or a Cobol programmer can understand them. Things like text editing, social simulations, control of household devices, animation and graphics, and communications networks.

Understandable Yet Challenging

As the premier magazine for beginners, it is our solemn responsibility to make what we publish comprehensible to the newcomer. That does not mean easy; our readers like to be challenged. It means providing the reader who has no preparation with every possible means to seize the subject matter and make it his own.

However, we don't want the experts in our audience to be bored. So we try to publish articles of interest to beginners and experts at the same time. Ideally, we would like every piece to have instructional or informative content—and some depth—even when communicated humorously or playfully. Thus, our favorite kind of piece is acessible to the beginner, theoretically non-trivial, interesting on more than one level, and perhaps even humorous.

David Gerrold of *Star Trek* fame says, "*Creative Computing* with its unpretentious, down-to-earth lucidity encourages the computer user to have fun. *Creative Computing* makes it possible for me to learn basic programming skills and use the computer better than any other source.

Hard-hitting Evaluations

At *Creative Computing* we obtain new computer systems, peripherals, and software as soon as they are announced. We put them through their paces in our Software Development Center and also in the environment for which they are intended—home, business, laboratory, or school.

Our evaluations are unbiased and accurate. We compared word processing printers and found two losers among highly promoted makes. Conversely, we found one computer had far more than its advertised capability. Of 16 educational packages, only seven offered solid learning value.

When we say unbiased reviews we mean it. More than once, our honesty has cost us an advertiser—temporarily. But we feel that our first obligation is to our readers and that editorial excellence and integrity are our highest goals.

Karl Zinn at the University of Michigan feels we are meeting these goals when he writes. "*Creative Computing* consistently provides value in articles, product reviews and systems comparisons . . . in a magazine that is fun to read."

Order Today

To order your subscription to *Creative Computing* send payment to the appropriate address below. Customers in the continental U.S. may call toll-free to charge a subscription to Visa, MasterCard or American Express.

Term	*USA*	*Canada and Foreign Surface*	*Foreign Air*
1 year	$20	$29 or £12.50	$50 or £21
2 years	$37	$55 or £24.00	$97 or £41
3 years	$53	$80 or £34.50	$143 or £61

We guarantee your satisfaction or we will refund your entire subscription price.

Join over 80,000 subscribers like Ann Lewin, Director of the Capital Children's Museum who says, "I am very much impressed with *Creative Computing*. It is helping to demystify the computer. Its articles are helpful, humorous and humane. The world needs *Creative Computing*."

creative computing

P.O. Box 789-M
Morristown, NJ 07960
Toll-free **800-631-8112**
(In NJ 201-540-0445)

27 Andrew Close, Stoke Golding
Nuneaton CV13 6EL, England

The magazine for Sinclair ZX80/1 users

SYNC

July/August 1981 Volume 1, Number 4

DEPARTMENTS

STRINGS AND ARRAYS

MATH AND MATH GRAPHICS

GAMES AND PROGRAMS

MACHINE LANGUAGE

INTERFACING

REVIEWS AND RESOURCES

COVER: Young beginners at Creative Computing's summer Computer Camp learning Basic operations on the ZX80 keyboard. Photo courtesy Morristown **Daily Record.**

Staff

Publisher/Editor-in-Chief	**David H. Ahl**
Editorial Director	**George Blank**
Managing Editor	**Paul Grosjean**
Associate Editor	**David Lubar**
Secretary	**Elizabeth Magin**
Production Manager	**Laura MacKenzie**
Typesetters	**Jean Ann Vokoun**
	Maureen Welsh
Financial Coordinator	**William L. Baumann**
Bookkeeper	**Patricia Kennelly**
Customer Service	**Ralph Loveys**
Order Processing	**Ruth Coles**
Circulation	**Frances Miskovich**
	Dorothy Staples
	Carol Vita

MEMBER MITA

Index to Advertisers

Volume 1, Number 4

SYNC is published bi-monthly for $10.00 per year by Creative Computing, 39 E. Hanover Ave., Morris Plains, NJ 07950. Second class postage paid at Morris Plains, New Jersey 07950, and additional entry offices.

Postmaster: Send address changes to SYNC, P.O. Box 789-M, Morristown, NJ 07960.

Subscriptions in USA: 6 issues $10; 12 issues $18; 18 issues $24. UK and foreign airmail subscriptions: 6 issues £10; 12 issues £18; 18 issues £25. Call (800) 631-8112 toll-free (in NJ, 201-540-0445) to begin your subscription.

Copyright 1981 by Creative Computing. All rights reserved. Reproduction prohibited in any form.

Glitchoidz Report

The GLITCHOIDZ REPORT will pass on to our readers errors, problems, and other Glitchoid activities which have been discovered. We welcome your contributions to this column.

GOTO Lines

Some readers have asked about a line such as GOTO 450 when the program does not have a line 450. In the ZX80 this does not stop the program or confuse the computer. The ZX80 will search for the line and, failing to find it, go on to the next line in the program after 450. In effect, it jumps over the GOTO command line in such cases.

Castle Doors (1:30)

Correct:
48 IF D=2 THEN LET A=RND(30)
62 IF D=5 THEN LET A$="ZOMBIE"

Draw a Picture (1:33)

Some readers have reported difficulties with this program, but it will run as printed. Pay especial attention to the last paragraph. When the program is entered and RUN, you are to respond to the prompt by entering the coordinates of the square you want to fill with a graphics symbol. You are then to respond to the prompt ENTER CHAR CODE with the number of the symbol you want as shown in column 3. Erasing may be done by entering 0 to erase the previous character or by entering the coordinates again and then the new character.

How to Produce a Display File
(2:13, col. 1)

A full screen suggestion:
40 FOR J=1 TO 21
Correct:
60 FOR I=1 TO 30
70 PRINT " "; (1 sp.)
(2:15, col. 1)

Continue adding dummy lines until line 10 is scrolled off the screen. Then delete the dummy lines by entering the line number and NEWLINE before entering the loader program.
(2:15, col. 2)
40 IF A<1 OR A>300 THEN GOTO 30

Truth in Programming (2:19, col. 1)
20 IF X THEN LET T=T+1

Game of Life (2:29)

Change: 450 NEXT I
Add: 460 FOR I=9 TO 55

The graphics given in the program do not produce the * in the square. Hopefully the * makes it easier to see the squares being referred to. For those who want to use the * in the display, change the character number 128 to 222 in lines 120 and 500.

Some readers have pointed out that the game does not follow the rules in the article. Reader Walter Bacon has proposed program changes to bring it closer to the rules (see Letters).

Artillery (2:27)

Readers with 1K have found that the game fills the memory rather quickly. You can increase the memory available for play by trimming down the PRINT statements to very brief prompts or try Reader Joe Dell'Orfano's program amendments (see Letters).

Tic Tac Toe (2:32)

Add: 445 CLS

More Truth in Programming (3:7, col. 3)

Correct:
IF NOT (X<5 AND Y<8) . . .

Black Hole (3:9)

Correct:
75 IF S<1 OR S>9 THEN GOTO 70

See the Letters for suggestions for trimming this to 1K.

Auto-Display-Changing (3:14)

Dr. Logan reports a bug that occurs because of variations in TV sets: "Some users may find it necessary to use other values in line 64.
64 POKE A+24,3 or 5
in order to get better timing."

Variable Conversions (3:26)

Correct:
113 IF X=-27 THEN GOTO 116

Forest Treasure (3:34)

Correct:
30 PRINT "■"; (shift A), or
30 PRINT CHR$(128)
490 delete %

8K Basic ROM and 16K-Byte RAM Pack (3:37)

As a number of readers observed, the actual ROM they received differs somewhat from our article. Between the time we received the materials and the actual production of the 8K ROM a few changes were made.

The following commands were omitted:

- DATA
- DRAW m,n
- NEW n
- READ v
- RESTORE
- UNDRAW m,n

These commands were added:

L PRINT	Prints a string on the printer.
L LIST	Lists the program on the printer.
COPY	Prints the entire screen on the printer. The printer is capable of printing all the characters including graphics.
FAST	On the ZX81 there are two modes of operation.
SLOW	On the ZX80 8K keyboard these do not function.

(3:38)

16K RAM Pack: A separate power supply is now being provided.

The ZX80 Keyboard
(3:42)

The author is David Ornstein, 25 Shute Path, Newton Centre, MA 02159.
(3:44)

Listing 1 is copyrighted by Sinclair Research.

Graphics Surprises (3:22)

The author is James H. Parsons, 1921 Flintlock Terrace West, Colorado Springs, CO 80918.

sync notes

Paul Grosjean

ZX81 — The Family Increases

A number of people have asked us whether we are going to include the ZX81 in our coverage. As our changed cover shows, the answer is a definite YES.

With the multiplication of the ZX80 family and the availability of new ROMs and RAMs, our readers want to know what the machine requirements of our programs are. If you are planning to send us an article or program that you have developed, be sure to state the minimum machine requirements for your program on a separate line below the title. When we are referring to the general family or discussing the family in areas where the ROMs and RAMs are not important, we will say simply the "ZX80," but where the ROM and RAM are important we want to include that information.

PERCEPTIONS

In this issue we are introducing the column PERCEPTIONS by David Ornstein (p.6). David has already contributed to *SYNC* through his work on the schematic of the ZX80 in issue 1, "Video Modifications for the ZX80" in issue 2, and "The ZX80 Keyboard" in issue 3.

Though he is Technical Services Manager for Sinclair Research Limited (U.S.), we must hasten to point out that the views expressed in his column will be strictly his own and in no way will represent Sinclair Research.

SYNCSUM

We are especially pleased with the first contribution of PERCEPTIONS to *SYNC* readers and authors in the concept and programs for the SYNCSUM. This is a method of checking whether you have entered the program correctly. If you are submitting an article, we ask you to include the SYNCSUM at the end of any program listing.

Spaces in PRINT Statements

Since we do not have a symbol on the ZX80 that marks clearly the empty spaces, sometimes problems arise in entering programs and getting them to work because the correct number of spaces in the PRINT statements is not always clear. When we receive a program done on a typewriter or a printer, we can usually figure out the number by counting the letters in the line above or below since each letter takes up the same amount of space. However, this is not always accurate because typographical errors can occur even in leaving spaces. Another problem comes up when we typeset programs. On the type setting machine letters vary in the amount of space they take up in the line and the spaces also vary in order to make the right hand margins even. So counting does not work. We have tried to handle this problem up to this point by indicating the number of spaces in a side note. Beginning with our next issue, we will be using a symbol to indicate spaces where these affect the running of the program.

If you are submitting a manuscript, we are asking you to *indicate all spaces* in the PRINT statements except the obvious ones between words by using the symbol #. We have chosen this because it is found on almost all typewriters, and it is not used on the ZX80 family of computers. We will use a different symbol in *SYNC* articles, but, even if we slip up, # will not cause any programming errors since it cannot be entered.

If in running a program you are sure that you have entered it correctly but it still does not work, check the number of spaces in any PRINT statements. You might experiment by changing these.

MicroAce II????

Contrary to some reports, MicroAce is not planning at the present time to offer a MicroAce II. When the present stock of kits is exhausted, the MicroAce computer will no longer be available from MicroAce. The company goal will be to offer equipment to upgrade the machines already sold. An 8K ROM and a flicker free video board (which requires 8K ROM) will be available by the time you receive this issue of *SYNC*. The RAM capacity will be expandable by two options: a 4K RAM and a 16K RAM. These are planned for the fall.

SYNC Subscribers Pass the 6000 Mark

At the end of June our subscription list totaled 6135 subscribers with 1532 outside the U.S. California leads the list with 13% of the total, followed by New York with 7%, and then by Illinois, Texas, and Massachusetts with about 4% each. Outside the U.S. the United Kingdom leads the list.

A P.S. from Alger Salt

Readers of Alger Salt's "A Parallel Interface for the ZX80/MicroAce" should note the following P.S. which arrived after our layout was completed:

After reading the article "The ZX80 Keyboard" in the May/June issue of *SYNC* I learned that the keyboard is an input device that is addressed by any even address. This accounts for the difficulty I encountered when trying to read from port A on the PIO. The problem can be avoided by using address line A2 instead of A0 to select the A or B port. That is, connect the A/B SEL line on the PIO to a different address line than A0. Then use only odd addresses when addressing the PIO, i.e., addresses with A0 = 1.

letters

Gauntlet and USR(47)

Dear Editor:

My *Gauntlet* program in your May/June issue can be greatly improved by using the technique described on page 6 of the same issue. On that page, Hasse Taube says that USR(47) will return the end of variables address. But the end of variables is right next to the start of the display file. That makes my machine language routine unnecessary.

The machine language routine made entering the program difficult and listing the program dangerous. But if you change line 900 to: 900 LET D=USR(47)+2, then you can ignore the subroutine loader and decimal listing and also delete line 1.

This is another example of how *SYNC* helps get the most out of the ZX-80. My thanks to Michael Kirkland, Hasse Taube and, of course, to *SYNC*.

Ken Berggren
104 Ridgeway Ave.
Louisville, KY 40207

*Widget*economics

Dear Editor:

. . . I greatly enjoy your magazine; however, I have found a few problems. In running the *Widget* program I have never been able to even break even. It is a challenge just to keep from going bankrupt. Is the program listing just advertisement?

Another thing I would like to see in your future articles is how to convert either mechanically or through a machine code subroutine the screen to active instead of going blank during runs.

Richard McDaniel
PO Box 71
Glasgow, VA 24555

Ed.—A number of readers of Widget *have found it quite challenging. See Reader Bacon's letter below. It seems that to remain solvent and become a successful capitalist, you must make some minor program changes.*

The conversion you ask for would cost more than the original computer, according to our sources. So it does not seem practical at this point.

Dear Editor:

. . . *Widget—NEAT PROGRAM*! No mistakes, but it is impossible (mathematically) to ever progress from those starting conditions. You can only minimize your losses to last as long as possible before bankruptcy overtakes you. The game is a good challenge if you start with *two* plants (or other assets like inventory) . . .

Game of Life. You printed a program from *Thirty Programs for the Sinclair ZX80 1K*. The book Addendum Page makes corrections in lines 450 and 460 (*See Glitchoidz Report*). However, even with these corrections the program *does not follow the logic rules* although it does run. To change the program so it follows the rules make the following corrections:

```
320 IF ((I+I/7) AND 7)=0 THEN GO
    TO 350
340 IF ((I-1+(I-1)/7) AND 7)=0 THEN
    GO TO 360
360 FOR J=6 TO 8
400 IF (J*R=-8 OR J*R=6) AND ((I-1+
    (I-1)/7) AND 7)=0 THEN GO TO
    420
405 IF (J*R=8 OR J*R=-6) AND ((I+I
    /7) AND 7)=0 THEN GO TO 420
```

If you do this and the publisher's changes in 450 and 460, the program will follow the rules in the article.

Walter W. Bacon
RR 7, Box 68
Hopewell Junction, NY 12533

*Ed.—For those who cannot abide by the decisions of the free market place and face bankruptcy fearlessly, a bit of Widgeteconomics can be performed by tinkering with the program to improve the market position. According to my program advisers, increase P in line 10 and/or change line 640 to read LET M=M-20*P-15. You can also try a number lower than 15.*

Artillery and *Black Hole*

Dear Editor:

The *Artillery* game depletes my 1K of RAM after about 5 or 6 turns. After searching through the listing for a mistake in my typing, I came up with the following changes:

Omit: Line 140

Change line 320 to: 320 GO SUB 150

Add: 80 DIM S(21)

The program should now work without any difficulties.

Also, the program *Black Hole* by Bill Eckel will run in 1K of RAM if the following changes are made:

Omit 5, 14, 16, 18, 30, 32, 34, 1025, 1130 & 1140

```
 100 LET B=S+1-2*(S/2)+2*(S/6)
     +4*(S/8)-3*(S/9)
 110 GO SUB 980
 120 LET B=S+3-2*(S/2)+(S/3)+3*
     (S/4)-4*(S/5)+5*(S/6)-5*(S/7)
     +2*(S/8)-5*(S/9)
 130 GO SUB 980
 140 LET B=S+4-6*(S/2)+5*(S/3)-
     (S/4)+5*(S/5)-6*(S/6)+7*(S/7)
     -2*(S/8)+2*(S/9)
 150 GO SUB 980
 160 LET B=0
 170 IF S=5 THEN LET B=8
 180 GO SUB 980
 960 GO TO 25

1075 IF X(I+5)=0 THEN RETURN
```

Or, as David Lubar might want to write it:

```
100 LET B=-2*(S=1)-(S=2)-2*(S=3)
    -(S=4)-2*(S=5)-3*(S=6)-4*(S=7)
    -7*(S=8)-5*(S=9)
120 LET B=-4*(S=1)-3*(S=2)-5*
    (S=3)-7*(S=4)-4*(S=5)-9*(S=6)
    -5*(S=7)-9*(S=8)-6*(S=9)
140 LET B=5*(S=1)-6*(S=3)-6*
    (S=5)-8*(S=7)-8*(S=9)
```

Replace lines 160 and 170 with:

```
160 LET B=-8*(S=5)
```

Lines 100,130,150, and 180 remain as in the first change.

As you can see, David Lubar's article about Boolean operations has been used to a great extent in shortening *Black Hole*.

Joe Dell'Orfanio
122 Weaver St.
Greenwich, CT 06830

Dear Editor:

Bill Eckel's *Black Hole* (*SYNC* 3:8) can be compacted to fit in less than 1K of memory, thus making it fit neatly into an unexpanded ZX-80. The following conversions should work:

```
Changes: 20   LET X(5)= -1
         44   IF X(I) THEN PRINT
              "*";
         46   IF NOT X(I) THEN
              PRINT 0;
         78   IF NOT X(S) THEN GO
              TO 70
         980  IF NOT X(B) THEN GO
              TO 986
         983  IF X(B) THEN LET
              X(B)=0
         986  LET X(B)= -1
         1010 IF X(I) THEN GO TO
              1050
         1050 IF X(5) THEN RETURN
         1070 IF NOT X(I) THEN
              RETURN
         1100 IF NOT X(I) THEN
              RETURN
         1120 PRINT "YOU WIN"
```

I used the first conversion as it was my own, and I hadn't yet read Mr. Lubar's article. I have found no problems in the playing of *Black Hole* after the conversion was made. I have also found that any one of the possible solutions is actually two solutions... just reverse the order. Happy Star-Shooting.

Mark Kleinman
4228-D FCN
McGuire AFB, NJ 08641

Basic Computer Games on the ZX80

Dear Editor:

Please tell me if the programs in *Basic Computer Games* and *More Basic Computer Games* work on the Sinclair ZX80 and VIC-1001.

A. Samereu
4946 Dornal
Montreal, Quebec H3W-1W2
Canada

Ed.—These programs will not work directly on the ZX80 for two reasons. First, they must be translated, that is, adapted to the specific form of Basic that the ZX80 uses. This is not difficult after you get some experience in programming and after you put SYNC *articles to work for you. Second, many programs even when translated will require more than 1K RAM. So before you enter a program, you can give it a rough check for size by comparing it to a 1K program printed in* SYNC. *If you want to be more precise, you can count the bytes in the translated program. The line entry requires 2 bytes; each keystroke in the line content counts as one; the NEWLINE entry adds one more.*

LED Fringe Benefit

Dear Editor:

I added the LEDs to monitor tape input as described by Cecil Bridges in the initial issue of *SYNC*. An additional advantage of this modification that he did not mention is that it eliminates the need to disconnect the ear cable on the recorder in order to position a tape for program loading (if you have a tape recorder with a digital counter). Simply advance the tape to the appropriate number on the counter, type LOAD, start the recorder, and when the red light goes out type NEWLINE.

William H. Caskey
1112 Pake Lane
Morris, MN 56267

Memory Mapping

Dear Editor:

The one thing I'd really like for your editors to address is how we get information into the ZX80 from the outside world using memory mapped input. I'm afraid to use the same approach for input that I used for output (i.e., writing to a nonexistent ROM address) because when I PEEK(4097) I get 64 decimal. This implies to me that somebody is on the data bus (at least D5). I'm afraid to put anything on the data bus for fear of having two chips on the bus at the same time and damaging someone. Of course I don't have a circuit schematic with the ZX80 so I can't really decide whether or not the risk exists or whether D5 just appears high because one tristate doesn't clamp all the way. Can you sell me a schematic for the ZX80?

William Byrne
2 Cypress Dr.
Wichita, KS 67206

Ed.—A suggestion from David Ornstein: Put the memory map input port in any address between 12 and 16K.

Schematics of the ZX80 are available at no charge from Sinclair Research Limited, 50 Staniford St., Boston, MA 02114.

SYNC Coverage

Dear Editor:

. . . I hope that all the new products coming out will not affect your policy of sticking to the basic machine. Anyone with new ROM can, I think, easily translate old ROM programs, whilst the converse is not always possible. I hope that you could follow the ideas of *Interface* the National ZX80 Users Club magazine over here and produce mainly 1K programs with an occasional 4K or more.

One thing I would like to see in *SYNC* is more attention given to PEEK, POKE, and USR. Most people can devise a program just using Basic, but if you have no knowledge of machine code or the Z-80, such as Ken MacDonald's EXCELLENT space intruders program advertised p. 19 [*issue 1*], are unintelligible—all I know is that they work . . .

I hope the ZX80 . . . catches on over in the States as well as it has here; if the example set by the TRS-80, PET and APPLE is anything to go by then we're in for a good deal of excellent American software—especially from Creative Computing!

Richard J. Barton
12 Mill Lane
Camblesforth,
Selby
North Yorks
YO8 8HW
U.K.

Ed.—While the scope of SYNC *must grow to meet the needs of our readers as they also grow in knowledge of the machine and expand their equipment, we will not leave behind the people with the basic ZX80 nor those people who are new to computing. Again, authors take note. PEEK, POKE, and USR are among the most frequently requested topics for articles.*

Currently Technical Services Manager, David Ornstein has been with Sinclair since the opening of its U.S. office. He has been involved with Sinclair's technical hotline, technical writing, and machine servicing. His primary interests are in the areas of software and hardware R & D, and system integration.

His secondary interests are reading (Frank Herbert), listening (Pink Floyd) and sharing

perceptions

David Ornstein

SYNCSUMs

One day, I was typing a system-check program into our computer. I took four and a half hours to enter the program. As I was about to run it, an awful thought occurred to me: What if I had made an error in my typing? Since the program had access to all parts of the system, a typo could be fatal. I decided to check it against the listing . . . once. Then I ran it. The end result—that I overwrote the system disk—is irrelevant. But what is important is this: *If the program listing had included the program's SYNCSUM*, I would have known better.

What is a SYNCSUM? A SYNCSUM is what is known as a checksum, or, rather, a modified version of a checksum. The checksum is a method of checking to see whether a program has been entered correctly by letting the computer add up all the bytes in a program. To use this error-checking method, you simply compare the checksum of the original program with the checksum of the program you have entered. If the numbers are not the same, you have made an error in entering the program. If the numbers are the same, the chances are about 90% that you have entered the program correctly.

In the ZX80, a certain area of memory is used to hold the current program. This area begins where the area for system variables ends. For the 4K ROM, this address is 16424 decimal (4028 hex); for the 8K ROM, 16509 decimal (407D hex). A system variable points to the first byte after the program area. The 4K and 8K ROMs format memory differently. In a 4K system, therefore, this variable is VARS (which points to the first byte of the VARiable Storage area), but in an 8K system, it is D-FILE (which points to the first byte in the Display FILE). These variables are stored at locations 16392 (4K) and 16396 (8K), respectively.

The assembly language program, shown in listing 1, is used to generate the current program's SYNCSUM on a 4K system. The corresponding program for an 8K system is shown in listing 2. You will notice that it is not adding all the bytes, but XORing them together. This is the modification of the standard checksum method referred to earlier. You will end up with a number which is less than 256.

To use the SYNCSUM program on a Basic program requires that the SYNCSUM program be resident (i.e., in memory) all the time. This can be accomplished first, by reserving some memory (RAM) such that Basic will not tamper with it, and, second, by loading the SYNCSUM routine into this area. Listings 3 and 5 are programs to reserve the required amount of memory, 27 bytes. They should be used on 8K and 4K systems respectively. Listings 4 and 6 are programs to load the machine language SYNCSUM generation program into this previously reserved memory space. These programs should be run at the beginning of any session of computer use when you may want to know a programs's SYNCSUM. From the time they are run until the computer is turned off, obtaining the SYNCSUM is simple: type

Label	Hex	Assembly Code	Comments
4KSSUM:	212840	LD HL,16424	;HL=Start
	ED5B0840	LD DE,(VARS)	;DE=Stop
	0600	LD B,00	;B=00 (Result Accumulator)
LOOP:	7C	LD A,H	;If HL=DE then XORNXT
	BA	CP D	
	2008	JR NZ,XORNXT	
	7D	LD A,L	
	BB	CP E	
	2004	JR NZ,XORNXT	
DONE;			;else done
	68	LD L,B	;low byte returned is SYNCSUM
	2600	LD H,00	;high byte is 00
	C9	RET	
XORNXT:			;XOR the next byte into the ;Result Accumulator.
	78	LD A,B	;Get current RA
	AE	XOR (HL)	;XOR it in
	47	LD B,A	;put back result into RA
	23	INC HL	;bump pointer
	18EE	JR LOOP	;go back for next byte

Listing 1.

PRINT USR(x), where x = your memory size (for example, 1024, 2048, 16384) - 27 + 16384, followed by NEWLINE as always. Thus x will equal 17381 for 1K, 18405 for 2K, and 32741 for 16K.

Enter (or LOAD) the RSV program (listing 3 or 5) and then RUN and NEWLINE. The 4K program will prompt you for "MEMORY SIZE?" Enter your memory size (1024, 2048, 16384) and NEWLINE. Next enter (or *on 8K systems only*, LOAD) the LDR program (listing 4 or 6). On a 4K system LOADing the LDR program after using RSV will cancel the effects of running the RSV program.

Press RUN and NEWLINE. Again the 4K program will prompt you for "MEMORY SIZE?" Again enter it and NEWLINE. The 4K "MEMORY SIZE?" prompt will return to the screen, but hit NEWLINE and you will return to program mode. The SYNCSUM routine is now resident.

On an 8K system, type NEW and NEWLINE. On 4K systems, as noted earlier, using the NEW command will delete the SYNCSUM routine from memory. Therefore, to clear out the 4K LDR program, you must delete each line individually. To delete, e.g., line 10, type 10 and NEWLINE. Repeat until the whole program is gone.

You can now begin entering *your* program. Once again, if you have an 8K system, you can LOAD your program. With 4K you must type in each line individually, as LOADing will destroy the SYNCSUM routine. You can now obtain the SYNCSUM at any point along the way via the PRINT USR (x) command (see above for the size of x). When you have finished and you are sure your program is correct, call for the SYNCSUM for the entire program. Write it down at end of your program for future reference. Be sure to include it after the end of any programs submitted to SYNC.

This method will also work just as well with the ZX81 since it uses the 8K ROM. I hope this idea is as helpful to ZX80 owners as it is to the rest of the computer world.

Until next issue, same relativistic time period, same non-euclidian universe. ■

Label	Hex	Assembly Code	Comments
8KSSUM:	217040	LD HL,16509	;HL=Start
	ED5B1240	LD DE,(D-FILE)	;DE=Stop
	0600	LD B,00	;B=00 (Result Accumulator)
LOOP:	7C	LD A,H	;If HL≠DE then XORNXT
	BA	CP D	
	2008	JR NZ,XORNXT	
	7D	LD A,L	
	BB	CP E	
	2004	JR NZ,XORNXT	
DONE:			;else done
	68	LD C,B	;low byte returned is SYNCSUM
	0600	LD B,00	;high byte is 00
	C9	RET	
XORNXT:			;XOR the next byte into the
			;Result Accumulator.
	78	LD A,B	;Get current RA
	AE	XOR (HL)	;XOR it in
	47	LD B,A	;put back result into RA
	23	INC HL	;bump pointer
	18EE	JR LOOP	;go back for next byte

Listing 2.

```
10 LET R=27  [the number of bytes to reserve]
20 LET RAMTOP=PEEK(16388)+PEEK(16389)*256-R
30 POKE 16388,RAMTOP-256*INT(RAMTOP/256)
40 POKE 16389,INT(RAMTOP/256)
50 NEW
```

Listing 3: 8K ROM RSV.

```
10 REM 217040ED5B124006007CBA20087DBB
   2004680600C978AE472318EE
20 LET RT=PEEK(16388)+PEEK(16389)*256
30 FOR B=0 TO 26
40 LET X=((PEEK(16509+5+B*2)-28)*16+(PEEK
   (16509+5+B*2+1)-28))
50 POKE RT+B,X
60 NEXT B
```

Listing 4: 8K ROM LDR

```
 5 LET R=27
10 PRINT "MEMORY SIZE ?"
20 INPUT M
30 LET M=M-R+16384
35 POKE 16999,33
40 POKE 17000,M
50 POKE 17001,M/256
60 POKE 17002,195
70 POKE 17003,107
80 POKE 17004,2
90 LET K=USR(16999)
```

Listing 5: 4K ROM RSV

```
10 REM  212840ED5B084006007CBA20087DBB
   2004682600C978AE472318EE
15 LET R=27
20 PRINT "MEMORY SIZE ?"
30 INPUT M
40 LET RT=M-R+16384
50 FOR B=0 TO 26
60 LET X=((PEEK(16424+3+B*2)-28)*16+(PEEK
   (16424+3+B*2+1)-28))
70 POKE RT+B,X
80 NEXT B
```

Listing 6: 4K ROM LDR

Machine Code Keyboard Scanning Program

Bernard Puerzer

Visions danced in my head! Visions of a completely controlled Amateur Radio station.

Imagine! A microprocessor-controlled system that would translate a Morse Code message and display it on the monitor, translate a message into Morse Code and transmit it at a pre-determined code speed, control a rotor to allow a beam to follow the Oscar satellites (satellites developed by an international Amateur Radio group for its exclusive use), and automatically log all the stations that I had communication with. The possibilities are endless.

I explained to my wife that a personal computer could do more than play games. (How else was I to persuade her that a computer was a necessary purchase?) I listed all the useful functions. At first she listened to my pipedreams in disbelief, but as my plans grew more detailed and I gave rational explanations of how my ideas could be accomplished, she became interested, then impressed.

My immediate interest in a computer was to develop a Morse Code transmit-receive converter. Then, when more memory became available, it would grow into the self-contained control system I had always envisioned.

My MicroAce 2K kit arrived, and it took less than a week to assemble. Finally I was ready to program a task. But wait, a good functional check of the system was in order. Why not program a few games? *Bombardment* is fun, and *Depth Charge* adds even more challenge. The new issue of *SYNC* contains an enjoyable behind the *Castle Doors* game, and I have got to try the "Draw a Picture" program. Two months later enough games had been played on the system to functionally check an IBM 370.

"Okay," My wife said. "You were right. Personal computers are useful. I mean, if you're ever stuck in a dungeon, at least you'd know which door to choose."

I chuckled at this but realized she had a point. Computer games can be a trap. My initial project was being ignored. It was time I got busy.

I felt that the Morse Code transmitter portion of my project would be the easiest since I would need additional circuitry if I wanted to receive Morse Code and translate it. The computer, as it stands, is not equipped to convert the output from the receiver into a digital waveform. Due to obvious memory constraints, the program has to be done in machine code. My plan was to read the keyboard input, find the input by consulting a look-up table, then convert it to a Morse Code type digital output which would clock a relay (on the transmitter). A software keyboard buffer is needed to allow the operation to 'type ahead' of the transmitted output, and a driver subroutine is needed to clock the relay at the desired code speed. The typed input should also be displayed on the monitor. This can be accomplished within the MicroAce 2K memory if the code is written properly.

The first stumbling block came when I tried to read the keyboard input, using machine language code.

The Problem

To read the input in Basic, an input statement is used, but the INPUT command looks for either a number or a letter and cannot be used to accept both randomly. I could not use it for my application, and other programs may have a similar problem. To read a key on the keyboard using one Basic routine would use up too much memory, and I doubt that it is even possible. Therefore, it is machine code all the way!

Even a person with no interest in Morse Code could find the keyboard input routine interesting since it has many other applications. If nothing else, it provides a better understanding of the Sinclair/MicroAce hardware.

The Solution

I began by studying the schematic to understand how the keyboard is read by the software. As it turns out, both the Sinclair and MicroAce use the same technique. The keys are wired in a matrix configuration, and the rows of the matrix are connected to the Z-80 address lines A8-A15, while the columns are connected to the Z-80 through a latch that is energized by the KEYBD signal (active whenever an I/O instruction is executed). When a key is depressed, the address line for the row of the key and the data line for the column of the key are connected. If the address line is low at that instant, the data line will be pulled low. Therefore, the Z-80 will analyze the data lines after a known address is issued with an I/O instruction to determine if a specific key was depressed.

If all the address lines A8-A15 were low, the Z-80 could not tell which row caused the data lines to change. So to scan the keyboard, each I/O instruction must have only one address line low at a time to determine the exact key that was depressed.

The Machine Code Monitor

Understanding the technique, I proceeded to write the machine code to decode the keyboard. Although I tried to keep the code as efficient as possible, it is still almost 100 bytes of instructions. Typing in this many POKEs did not seem much of a challenge so I wrote the following program in Basic:

```
  5 LET MARK=0
 10 PRINT "ENTER STARTING
      ADDRESS"
 20 INPUT A
 30 PRINT A; " (1 sp.) "; PEEK (A)
 40 INPUT B
 60 IF B    255 THEN GO TO 130
 70 POKE A,B
 80 PRINT A; " (1 sp.) "; PEEK (A)
 90 LET MARK=MARK +1
100 IF MARK    10 THEN GO TO 110
105 CLS
107 LET MARK = 0
109 PRINT A; " (1 sp.) "; PEEK (A)
110 LET A = A+1
120 GO TO 30
130 STOP
```

Figure 1.

Bernard Puerzer, 3209 So. Kinnickinnic Ave., Milwaukee, WI 53207.

This program allows easy loading of sequential memory locations. When it begins, it asks for the starting address, entered in decimal. The program then displays the address and its current memory contents.

Enter the new contents in decimal, followed by a return. The program displays the new contents and then automatically increments to the next location, displaying the address and current contents. Continue entering your machine code program. When that is completed, enter a number larger than 225 to stop the monitor program.

I am sure you will find this method much easier to use than entering a POKE instruction for every machine code instruction to be loaded. Since so much machine code is written in hexidecimal notations in the 'real world,' a good modification to this monitor would be to allow the memory contents to be loaded by entering the number in hex notation. Since this would require entering the numbers (0-9) and letters (A-F), a keyboard input program such as the one to be described would be required. Now that we have an easy way to enter machine code on our Sinclair/MicroAce the rest is a piece of cake.

I have not yet devised a clean way of saving long machine code programs on a cassette tape, but I did find a technique that works. If you set up a few DIM statements in the beginning of the monitor program to dimension a few arrays with variables not used in the program, the system will 'reserve' memory locations for these arrays. The machine code can now be loaded, and, when the SAVE command is executed, the Basic program, including the arrays, will be saved. With some luck, the machine code program will reside in the 'reserved' space and be saved. When the program is downloaded from tape and re-run, be sure to use the GO TO and not the RUN instruction to begin the program, or the array space will be erased.

Keyboard Input Program

The program, as written, resides in memory locations 17401-17497. If this is not convenient on your system, the program can be re-located easily enough by changing a few of the instructions that reference memory.

When the program is run, the code of the depressed key is placed in memory location 17400. Therefore, a PRINT CHR$ (PEEK(17400)) command will display the key depressed on the keyboard. Other uses of this code can be devised.

As the program is being executed, it will loop between addresses 17405-17412 until a key is depressed. The INA,(C) instruction will place the contents of the B Register onto the address lines A8-A15 and the contents of the C Register onto address lines A0-A7. Therefore, by rotating a zero through the B Register and keeping the C Register set to zero, the keyboard can be scanned. When the data lines change, we know a key was depressed. Now we must decode the findings.

When a key is depressed, the accumulator and the B Register are analyzed to determine which key was depressed. Figure 2 shows the A and B Register contents for each key.

Register B Contents	Accumulator Contents 30	29	27	23	15
254	shift	Z	X	C	V
253	A	S	D	F	G
251	Q	W	E	R	T
247	1	2	3	4	5
239	0	9	8	7	6
223	P	O	I	U	Y
191	new line	L	K	J	H
127	space	•	M	N	B

Figure 2.

LAMO-LEM PRESENTS:

A NIGHT IN LAS VEGAS

FOUR COMPUTER GAMES FOR THE ZX80 AND MICROACE.

CRAPS

BET THE PASS LINE (WITH ODDS), HARDWAYS, THE FIELD, BIG 6 & 8, ETC., (WITH RULES). USES LAS VEGAS ODDS.

ROULETTE

NUMBERS FLASH PAST UNTIL A WINNER COMES UP. BET THE NUMBERS, ODD/EVEN, RED/BLACK, O, OO, COLUMNS, DOZENS, ETC . . .

BLACKJACK

USES FOUR DECKS. SHUFFLES AUTOMATICALLY. PLAY AGAINST THE DEALER. BLACKJACK WINS 1½-1. GRUBSTAKE KEY WHEN YOU GO BROKE!

SLOT MACHINE

PULL THE HANDLE (NEWLINE), AND WATCH THE REELS CLICK INTO PLACE. ASSORTED PAYOFFS. WITH FULL GRAPHICS.

ALL FOUR GAMES ON ONE CASSETTE, WITH MANUAL, MINIATURE LAYOUTS, CHIPS, AND FULL COLOR KEYBOARD OVERLAYS. FOR ENTERTAINMENT ONLY. REQUIRES 4K BASIC AND 1K MEMORY OR MORE.

ALSO FROM LAMO-LEM: THE CHEST OF CLASSICS. FOUR CLASSIC COMPUTER GAMES. LIFE, LUNAR LANDER, MINDMASTER, AND K-TREK (WITH PHASORS, SCANNERS, VIEW SCREEN, & WARP DRIVE). CASSETTE, MANUAL, CHARTS, OVERLAYS, & MORE. 4K ROM, 1K RAM, $9.95.

$9.95

SEND FOR OUR CATALOG OF ZX80, MICROACE, APPLE, AND TI 57, 58, & 59 PRODUCTS, INCLUDING FREE ZX80/M.ACE CODING SHEETS.

**NO POSTAGE.
NO HANDLING.
NO SALES TAX.**

LAMO-LEM LABS

CODE 204, BOX 2382, LA JOLLA, CA 92038

The program now checks each bit of the accumulator to determine which one is low. Register C is incremented once for each bit tested. Then each bit of Register B is checked to determine the one that is low. Register C is incremented by five for each bit tested.

The contents of Register C are then used as an offset for the look-up table found at addresses 17458-17497. The look-up table value is placed in location 17400 and the subroutine returns to the program that called it.

It should be noted that this program reads the total keyboard but only 'lower-case.' Code could be added to look for the SHIFT key code. If detected, the program could then add an offset to Register C and jump back to look for another key to be depressed. The look-up table must then be expanded to include all the SHIFT characters.

Conclusion

Although the code may be difficult to follow at first, the program is really doing a lot in 97 bytes of memory.

By the way, this program is fast! If you use it as a subroutine in a Basic program, be sure enough Basic instructions precede the USR instructions, or the keyboard input program will decode the NEWLINE key that you depress after the RUN instruction—unless, of course, you release it in a matter of milliseconds. A good trick is to include a FOR loop of about 10 just before the USR call instruction to give you enough time to release the NEWLINE key.

Now that I can read the keyboard, it is just a small matter of time before hitting the ham bands. But first, maybe I had better functionally check the system by running a few quick games of Acey Ducey.

Keyboard Scanning Program

Address	Decimal	Octal	Comments
17401	14	16	LD C,0
17402	0	0	
17403	6	6	LDB, 254
17404	254	376	
17405	203	313	RLC B
17406	0	0	
17407	237	355	INA, (c)
17408	120	170	
17409	254	376	CP A, 31
17410	31		
17411	40	50	JR Z,
17412	248	370	(-8) 2's compliment
17413	221	335	LD IX, Table addr -6(17452)
17414	33	41	
17415	44	54	
17416	68	104	
17417	95	137	LD E,A
17418	0	0	NOP] For future
17419	0	0	NOP] reference
17420	14	16	LDC,1
17421	1	1	
17422	22	26	LD D,0
17423	0	0	
17424	62	76	LDA, 00111011
17425	59	73	
17426	198	306	ADD A,000010000
17427	8	10	
17428	50	62	LD(17437),A
17429	29	35	low
17430	68	104	high
17431	103	147	LD H,A H is holder
17432	122	172	LDA,D
17433	129	201	ADD A, C
17434	87	127	LD D, A
17435	124	174	LDA,H
17436	203	313	BIT E
17437	X	X	Don't care
17438	32	040	JR NZ, cont.
17439	242	362	-14
17440	88	130	LD E,B
17441	203	313	BIT C, 2
17442	81	121	Test for a five
17443	14	016	LDC,5
17444	5	5	
17445	40	050	JRZ, back
17446	233	351	-23
17447	122	172	LDA,D
17448	50	062	LD(17453),A
17449	45	55	low
17450	68	104	high
17451	221	335	LD A,(IX+d)
17452	126	176	d
17453	x	x	don't care
17454	50	062	LD 17400, A
17455	248	370	low order
17457	201	311	high order
17456	67	103	high order
17457	201	311	RET
17458	1		table starts
17459	63		
17460	61		
17461	40		
17462	59		
17463	38		
17464	56		
17465	41		
17466	43		
17467	44		
17468	54		
17469	60		
17470	42		
17471	55		
17472	57		
17473	29		
17474	30		
17475	31		
17476	32		
17476	32		
17477	33		
17478	28		
17479	37		
17480	36		
17481	35		
17482	34		
17483	53		
17484	52		
17485	46		
17486	58		
17487	62		
17488	231		
17489	49		
17490	48		
17491	47		
17492	45		
17493	0		
17494	155		
17495	50		
17496	51		
17497	39		

End of Program.

The TL$ Function

Rolf L. Miller

Do not overlook the use of the TL$ function when you are creating programs. It is a very useful item. This function allows the ZX80 user to process a string in much the same way that other computers READ DATA statements.

To see how it works in this fashion, consider first the CODE(string) function. It will "read" and give the code of the first character in a string. Thus, if A$="ABC", CODE(A$) would result in 38, the code for A.

Now add the TL$ function: LET A$=TL$(A$). The TL$ function strips the first character from the string—A in this case—leaving "BC" in A$. If CODE(A$) is now reintroduced, it will "read" B and give its code, 39.

Clearly, an entire string can be "read" in this way. So, for example, say you have a stock portfolio of five stocks, namely: 100 shares of ABC, 200 of XYZ, 300 Q, 200 KLMN, and 100 ZX. The following program will print the number of shares in 100s, the stock symbol, ask for the last (current) price per 100 shares (stock prices are quoted per share so that 5 1/4 would be input as 525), and, then, after all five stocks have been processed, print the total value of the portfolio.

Rolf L. Miller, 492 S. Anacapa, Ventura, CA 93001.

"...It's a new game called 'Artillery'! Pretty Realistic, Huh...?"

```
 10 LET V=0
 20 LET P$=".1 ABC.2 XYZ.3 Q.2 KLMN.1 ZX."
 30 IF CODE(P$)=27 THEN LET P=CODE(TL$(P$))-28
 40 PRINT CHR$(CODE(P$));
 50 LET P$=TL$(P$)
 60 IF P$="" THEN GOTO 130
 70 IF NOT CODE(P$)=27 THEN GOTO 40
 80 PRINT, "  INPUT LAST PER 100"
 90 INPUT L                          —use 100 for test RUN
100 LET V=V+(L*P)
110 CLS
120 GOTO 30
130 CLS
140 PRINT "PORTFOLIO VALUE=  ";V  —test RUN should
                                              give 900
```

In line 20 it is noted that the code for . is 27 and acts as a flag to control the loop routine following.

In line 30, subtracting 28 from the code for 1, 2, 3, etc. results in 1, 2, 3, etc. since the code for 1 is 29, for 2 is 30, etc. and thus sets P at the proper value. Note here that the TL$ function is used to "look" one character ahead in the string without actually stripping the string.

To see the value of the TL$ function here, try writing a program without using TL$ to accomplish the same results as this program and look at the length and memory difference.

Another example of using TL$ is seen in this version of Mastermind. Further applications will be left to your imagination.

Mastermind

```
  5 REM COPYRIGHT 1981 BY ROLF L. MILLER
 10 LEX X=15
 20 LET A=9999+RND(22768)
 30 LET A$=STR$(A)
 40 LET B$=A$
 50 IF X=0 THEN GOTO 190
 60 LET X=X-1
 70 LET Z-0
 80 PRINT "5 NO.GUESS ";
 90 INPUT G$
100 IF G$=A$ THEN GOTO 220
110 PRINT G$;
120 IF B$="" THEN GOTO 170
130 IF CODE(G$)=CODE(B$) THEN LET Z=Z+1
140 LET G$=TL$(G$)
150 LET B$=TL$(B$)
160 GOTO 120
170 PRINT "  ";Z;" RIGHT (";X;")"  (2 sp.)
180 GOTO 40
190 PRINT
200 PRINT "YOU LOSE THE NO. WAS ";A$
210 STOP
220 PRINT
230 PRINT ,A$
240 PRINT
250 PRINT ,"YOU WIN"
```

Program Notes:

110 displays number guessed.

170 displays number of digits in proper place of sequence and number of turns left in ( ).

A Subroutine for Serial Data Output

S. Onsy

Trying to write machine code subroutines for my ZX80 presented some problems. This article details the problems with their solutions, and shows a simple subroutine to output data serially by bit to an asynchronous peripheral.

The first problem was to find a space in RAM to write my subroutine. The obvious space was the SPARE area shown in appendix II of the *ZX80 Manual* (See Figure 1). It is easy to find the start of the spare area by PEEKing into locations 16400 and 16401 which point to the display file end "DF-END" (*ZX80 Manual*, appendix III). But the problem is that the SPARE area is sandwiched between two dynamic areas. The VARIABLES, WKG SPACE, and DISPLAY FILE may expand pushing DF-END closer to the top of stack and overwriting my routine. The stack itself may get bigger and overwrites the routine.

A second problem came up when I tried to save the program on cassette. My machine code subroutine was not saved simply because the SAVE statement causes the ZX80 to save on cassette from the start of RAM up to E-LINE only (Figure 1).

A technique around these problems was to include my subroutine in a REMark statement and thus allocate a fixed area of RAM for it. I was able to save it on cassette, too.

S. Onsy, P.O. Box 2952 SAFAT, Kuwait, State of Kuwait, Arabian Gulf.

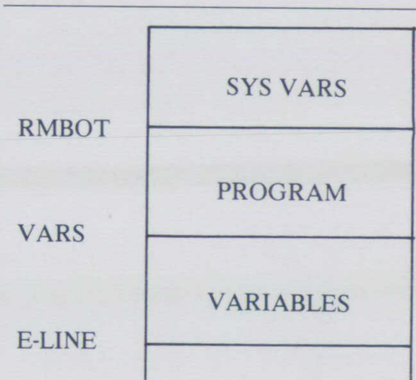


Figure 1. ZX80 Memory Map.

I used the following procedure to input the subroutine:

1. Calculate the length, in bytes, of the subroutine.

2. Enter a REM statement at line 1. Line 1 is used to insure that the REM statement will always be the first one in the program and that it will have a fixed address in RAM. Using numerics helped counting the number of reserved bytes. The REM statement appeared as follows:

0001 REM 01234567890123456789012345 67890 etc.

3. Enter the subroutine starting at address 16428.

Restrictions

While writing the subroutine I noticed the following:

1. Never use the code/data of 76 hex since it is an end of statement to the ZX80. OP code 76 hex is not used anyway since it is a HALT command to the Z-80 microprocessor.

2. Since codes 40 to 7F hex cause problems with the ZX80 LIST command if they are included in the REM statement, I avoided displaying the REM statement. This was achieved by adding dummy statements until it disappeared from the screen and then deleting the dummy statements.

Applications

The listed Z-80 subroutine was used to output data asynchronously to a serial printer at 300 baud. The output was taken from IC-11 pin 11 (Figure 2). The signal is at TTL level, and therefore the interface circuitry in the printer was bypassed (Figure 3). The baud rate can be changed by simply changing the bit time loop.

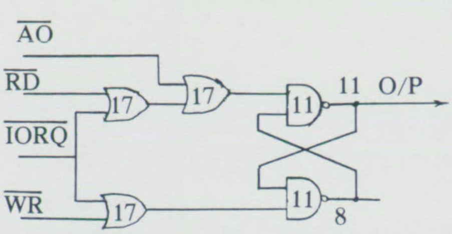


Figure 2. Serial O/P from IC11.

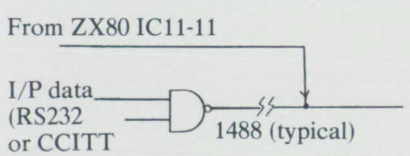


Figure 3. Typical I/F circuit.

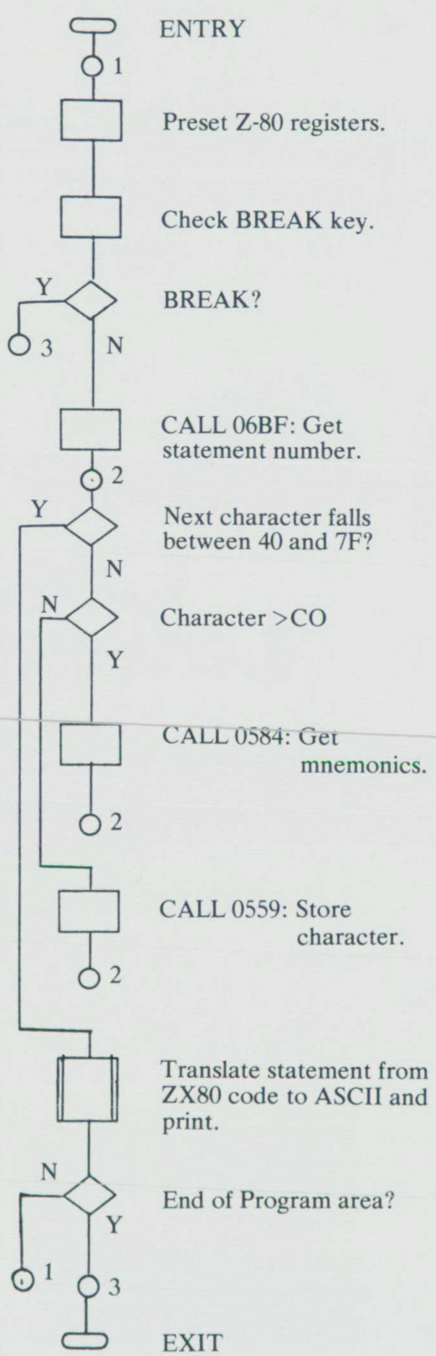
The Basic program shown uses the above subroutine to LIST itself on the printer from the PROGRAM area in RAM. The program is slow and not practical to use. However, it demonstrates some techniques for the ZX80. I have included enough REMarks to make the program self-explanatory. Since the program uses a flow similar to the ZX80 LIST command, it will be practical and much faster if the program is rewritten in the Z-80 code making use of the ROM subroutines. In the following discussion all addresses, codes, and data are in hex. The registers mentioned are the Z-80 internal registers.

The heart of the ZX80 LIST statement is a call to 04F7 which edits statement lines from the PROGRAM area in RAM into the display file. Register HL' points at the statement being edited while the resulting statement is stored at the location pointed at by DF-END. The 04F7 subroutine further calls two subroutines:

06BF: Translates statement numbers from binary to decimal expressed in the ZX80.

0684: Changes the commands and operators (i.e., codes > D3) to their proper mnemonics.

The following flow chart shows how a printer LIST routine that uses the ROM subroutines may look:



NEW ENGLAND SOFTWARE

7 GAMES FOR THE ZX80 AND MICROACE ON CASSETTE

MASTERMIND
DOUBLEMIND
SLOT MACHINE
CRAPS
TIC TAC TOE
SUB RESCUE
WHITE HOT NUMBER

ALL RUN IN 1K RAM

New England Software
Box 691
Hyannis, MA. 02601

$11.00 ORDERS MAILED First Class (U.S.A.)
£ 6.00 Air Mail (England)

YOUR BEST VALUE IN QUALITY SOFTWARE

Are you in SYNC?

If not, you should be. We would like any programs, translations of existing programs, games or tips which you have to pass on to fellow Sinclair ZX-80 or Micro-Ace owners. Articles are much more lively if accompained by photos (black and white), diagrams, and illustrations. If you do not have an output printer, please type program listings and carefully check them against the listing on the screen. Sample runs should be included with programs rather than just a description of what the program does. Articles should be typed, double space. Your name and address, with phone number should be on first page; all other pages should be numbered. All submissions should include return postage. Payment ranges from $15 to $40 per printed page.

Please send all submissions to:

SYNC
39 E. Hanover Avenue
Morris Plains, New Jersey 07950

LABEL	OP		OPERAND	ADR	CODE
CHRCTR	PUSH	HL	SAVE HL	402C	E5
	LD	H,(IY+11)	GET ADDRESS OF	402D	FD 66 11
	LD	L,(IY+10)	DF-END AND	4030	FD 6E 10
	INC	HL	INCREMENT	4033	23
	INC	HL	TWICE	4034	23
	BR	(HL)	GET NEXT BIT	4035	CB 1E
	POP	(HL)	RESTORE HL	4037	E1
	DJNZ	BITTEST	IF NOT LAST BIT GO TEST IT	4038	10 0C
STOPBIT	IN	A,(FE)	OTHERWISE OUT	403A	DB FE
	JR	BITTIME	A STOP BIT AND RETURN	403C	18 11
ENTRY	LD	HL,402C	INITIALIZE HL	403E	21 2C 3F 24
	SCF		AND PREPARE FOR	4042	06 09
	CCF		START BIT	4044	37
BITTEST	PUSH	HL	SET RET ADDRESS	4045	35
	JR	NC,SPACE	IF ZERO SPACE	4046	30 04
MARK	IN	A,FE	OTHERWISE MARK	4048	DB FE
	JR	BITTIME		404A	18 02
SPACE	OUT	(FF),A		404C	D3 FF
BITTIME	LD	D,80	START OF ONE BIT	404E	16 80
	LD	E,F6	TIME LOOP	4050	IE FD
LOOP	INC	D		4052	14
	JR	NZ,LOOP		4053	20FD
	INC	E		4055	IC
	JR	NZ,LOOP		4056	20 FA
	RET			4058	C9

Output Subroutine

This is a subroutine to output one byte serially by setting and resetting a latch in the ZX80. The data is preceded by a start and followed by a stop bit. The subroutine expects the byte to be at (DF-END)+2. All addresses, codes, and data are in hex.

Computer Lawnmower

Can a computer mow your lawn? Not yet. But a flowchart can show you how to make money cutting five lawns a day. The flowchart is easy. Mowing the lawns is still hard work.

Dr. Sylvia Charp and Marion Ball wanted a way to introduce basic computer concepts to children in grades 5 to 9 of the Philadelphia City Schools. So they identified some tasks that kids understood like mowing lawns, issuing paychecks and controlling traffic lights. They showed how computers are used in these tasks.

Flowcharts - A basic concept

They devised flowcharts. They located scores of photos. And they found an artistic high school student to illustrate these concepts with lively full-color drawings.

They then wrote a light-hearted but informative text to tie it all together. It talked about kinds of computers, what goes on inside the machine, the language of the computer and how computers work for us.

They took the problem of averaging class grades and showed how a simple yrogram could be written to do this job.

Well-qualified authors

Marion Ball has written other books on computer literacy. Sylvia Charp is the director of educational compuuting for Philadelphia City Schools. They pooled their talents to produce this book, Be A Computer Literate.

This easy-to-read book explains how computers are used in medicine, law enforcement, art, business, transportation and education. It's interesting and understandable.

Too much demand

The Bell System distributed 50,000 copies to schools throughout the U.S. but they couldn't meet the continuing demand. So Creative Computing Press now distributes the book. It's just $3.95 plus $1.00 shipping and handling. Send name and address plus payment or credit card number and expiration date to Creative Computing Press, Morris Plains, NJ 07950. Visa, MasterCard and American Express orders may also be called in toll-free to 800-631-8112 (in NJ 201-540-0445).

Order yours today. If, after reading it, you do not feel that you are "computer literate," return it for a full refund plus your postage to send it back.

creative computing

Morris Plains, NJ 07950
Toll-free **800-631-8112**
(In NJ 201-540-0445)

```
0001 REM  < CLEAR ) CLEAR (77f2 OR (NOT REM <)5GZ8  RZ<K NOT RE
M < N?=2 CLEAR *4 CLEAR 04 IF B89
0005 REM *****************************************
0010 REM * THIS PROGRAM LISTS ON AN ASCII SERIAL       *
0015 REM * PRINTER THE PROGRAM AREA OF THE ZX80 RAM.   *
0020 REM * THE REM STATEMENT 0001 CONTAINS A Z80       *
0025 REM * CODE TO OUTPUT ONE CHARACTER ASYNCHRONOUSLY*
0030 REM * AT 300 BAUD. THE ENTRY POINT IS AT ADDRESS *
0035 REM * 16446. THE ASCII CHARACTER SHOULD BE AT DF-*
0040 REM * END+2. THE OUTPUT TO THE SERIAL PRINTER     *
0045 REM * WILL BE AT TTL LEVEL AT PIN 11 OF IC 11     *
0050 REM * INTEGRATED CIRCUIT OF THE ZX80.             *
0055 REM * THE PROGRAM ALSO DEMONSTRATES THE USE OF    *
0060 REM * STRING VARIABLES AS A TRANSLATION TABLE TO *
0065 REM * GET ASCII CODES FROM ZX80 CODES.            *
0070 REM * THE PROGRAM USES A TABLE IN THE ZX80 ROM    *
0075 REM * USED BY THE LIST STATEMENT TO GET MNEMONICS*
0080 REM * FOR THE COMMAND/OPERATOR CODES.             *
0140 REM *****************************************
0150 REM *
0160 REM ***** TRANSLATION TABLE *****
0170 REM *
0180 LET X$="44644444444448UZCDHFEJXYWVGI"
0190 REM
0200 REM A=ADDRESS OF CHARACTER TO BE PRINTED. INITIALISED TO
16424
0210 LET A=16424
0220 REM LA=LAST ADDRESS TO BE PRINTED. SET TO VARS.
0230 LET LA=PEEK(16392)+256*PEEK(16393)
0240 REM ***** SUBROUTINE ADDRESSES *****
0250 LET CRLF=1070
0260 LET SPACE=1190
0270 LET PRINT=1240
0280 LET XLATE=970
0290 REM
0300 REM ***** MAIN PROGRAM *****
0310 REM INITIALISE LINE COUNTER
0320 LET LC=1
0330 REM NEW PAGE
0340 FOR J=1 TO 5
0350 GO SUB CRLF
0360 NEXT J
0370 REM       STATEMENT NUMBER
0380 LET SN=PEEK(A)*256+PEEK(A+1)
0390 GO SUB SPACE
0400 FOR J=1 TO 4
0410 LET T=SN/1000
0420 LET C=T+48
0430 GO SUB PRINT
0440 LET SN=(SN-T*1000)*10
0450 NEXT J
0460 LET A=A+1
0470 REM        STATEMENT
0480 LET A=A+1
0490 LET C=PEEK(A)
0500 IF C=118 THEN GO TO 880
0510 IF C>211 THEN GO TO 610
0520 IF C>27 THEN LET C=C+20
0530 IF C>57 THEN LET C=C+7
0540 IF C<28 THEN GO SUB XLATE
0550 GO SUB PRINT
0560 GO TO 480
0565 REM
0570 REM       COMMANDS AND OPERATORS
0575 REM
0580 REM F1=FLAG TO INSERT SPACE BEFORE
     AND AFTER MNEMONICS.
0590 REM F2=FLAG TO DEFINE END OF MNEMONIC.
0600 REM B =ADDRESS OF CURRENT CHARACTER
     IN MNEMONICS TABLE.
0610 LET B=186
0620 LET F1=0
0630 LET F2=0
0640 FOR J=0 TO C-212
0650 LET B=B+1
0660 IF PEEK(B)<128 THEN GO TO 650
0670 NEXT J
0680 LET B=B+1
0690 LET C=PEEK(B)
0700 IF C<128 THEN GO TO 730
0710 LET F2=1
0720 LET C=C-128
0730 IF C>27 THEN GO TO 760
0740 GO SUB XLATE
0750 LET F1=2
0760 IF F1>0 THEN GO TO 790
0765 LET D=C
0770 GO SUB SPACE
0775 LET C=D
0780 LET F1=1
0790 LET C=C+27*(1/F1)
0795 IF C=32 THEN LET F1=1
0800 GO SUB PRINT
0810 IF F2=0 THEN GO TO 680
0820 IF F1=2 THEN GO TO 480
0830 GO SUB SPACE
0840 GO TO 480
0850 REM
0860 REM      END OF STATEMENT
0870 REM
0880 GO SUB CRLF
0890 LET A=A+1
0900 IF LA>A THEN GO TO 920
0910 STOP
0920 IF LC>51 THEN GO TO 320
0930 GO TO 380
0940 REM
0950 REM *****     XLATE SUBROUTINE      *****
0960 REM
0970 LET Y$=X$
0980 FOR J=0 TO C
0990 LET Y$=TL$(Y$)
1000 NEXT J
1010 LET C=CODE(Y$)
1020 RETURN
1030 REM
1040 REM *****     CRLF SUBROUTINE      *****
1050 REM
1060 REM CC=CHARACTER COUNT
1070 LET CC=0
1080 LET LC=LC+1
1090 LET C=13
1100 GO SUB PRINT
1110 LET C=10
1120 GO SUB PRINT
1130 REM DELAY FOR CARRIAGE TO SETTLE
1140 FOR K=1 TO 5
1150 NEXT K
1160 RETURN
1170 REM
1180 REM *****     SPACE SUBROUTINE      *****
1190 LET C=32
1200 REM
1210 REM *****     PRINT SUBROUTINE      *****
1220 REM
1230 REM C=ASCII CODE FOR CHARACTER TO BE PRINTED
1240 LET CC=CC+1
1250 POKE PEEK(16400)+256*PEEK(16401)+2,C
1260 LET G=USR(16446)
1270 IF 73>CC THEN GO TO 1300
1280 GO SUB CRLF
1290 GO TO SPACE
1300 RETURN
```

How Is It Done?

Screen Scrolling

Dr. I. S. Logan

Introduction
This article shows how a routine can be written and entered into a ZX80 that enables the user to SCROLL the display. In the 4K monitor there is no facility at all for doing other than printing to the last line of the display, and then, when the display is full, the program will stop unless a CLS (clear screen) command is used.

The 8K monitor does have a SCROLL command, but it is limited in use as it only enables the user to scroll the whole display one-line-up and to print to the bottom line again.

The routine in this article will only work under the 4K ROM.

Objectives
My first objective was to produce a routine that would simply scroll the display one-line-up, when called by a USR command and allow the user to continue printing at the end of the display. However, a second objective soon appeared and that was to extend the routine so that only a predetermined part of the screen would be scrolled, thereby enabling the user to have a "title" area at the top of the display that would remain un-scrolled. The routine would require from the user that the number of lines to be left un-scrolled be specified, using a POKE command, before using the USR command.

Dr. Ian S. Logan, 24 Nurses Lane, Skellingthorpe, Lincoln LN6 OTT, England. This article is the third in a series.

The Theory
Before writing a routine that manipulates the contents of the display file, we must have a clear understanding of the structure of the display file of the ZX80. Figure 1 shows the parts of the display file as they would be produced by running the simple program:

10 PRINT "FIRST LINE"
20 PRINT "SECOND LINE"
Press RUN and NEWLINE.

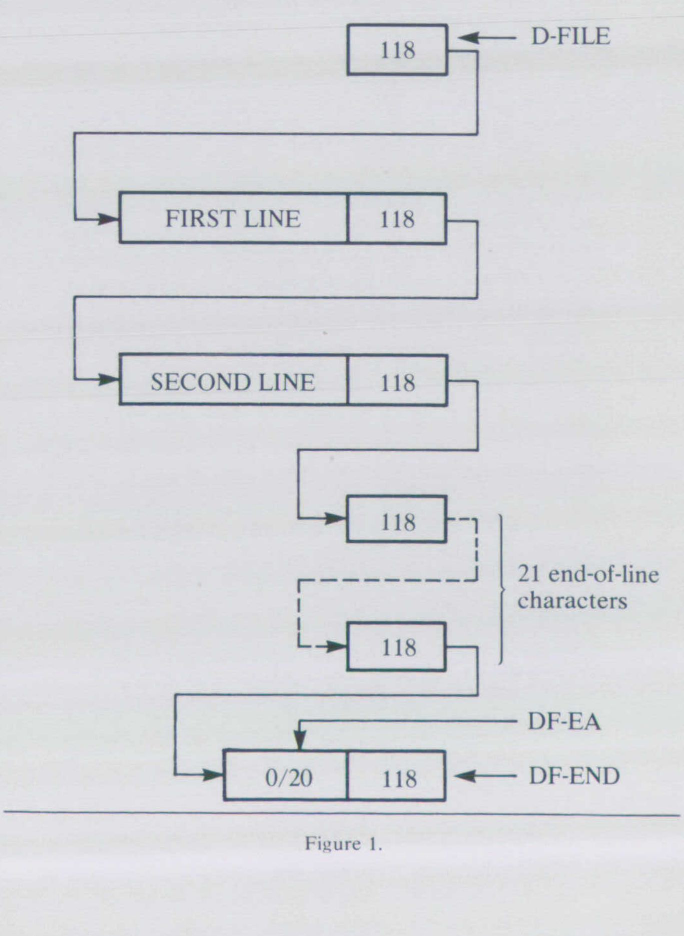


Figure 1.

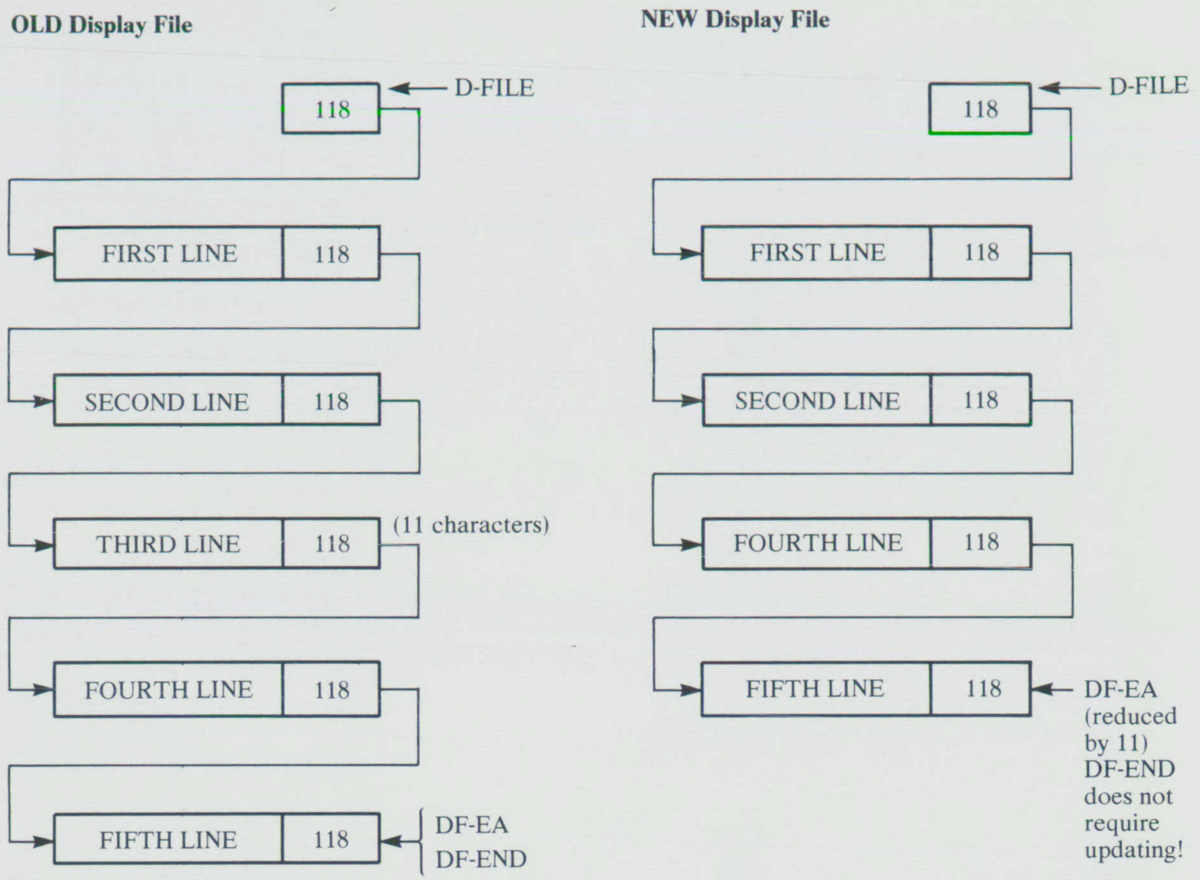


Figure 2.

Note that the lines are of varying length and that they all end in a "118" (Hex. 76) which is the end-of-line marker.

The pointer D-FILE always points to the first character in the display file, which is always an end-of-line character.

The pointer DF-EA points to the start of the "lower part of the screen," and DF-END points to the twenty-fifth end-of-line marker.

It is important to realize that the pointers DF-EA and DF-END are only given their final values when the execution of a program finishes. Before this time the pointers are being changed as each character is added to the display file. Therefore before the "end of program" routine is executed the pointers DF-EA and DF-END both point to the last location in the partially completed display file.

In a scroll routine it is necessary initially to collect the current value of D-FILE and then look through the display file until the point is reached that is to become the "new" contents of D-FILE. However, if certain lines are not to be scrolled, then these lines must be passed over, and the last end-of-line marker considered to be D-FILE.

The length of the line to be erased is then found, and the scrolling is achieved by moving the whole of the remainder of the display file down in memory so that it overwrites the scrolled line. There then remain two house-keeping tasks. The value of DF-EA and the value of the system variable 16421, the line counter, need to be altered. DF-EA has to be reduced by the 'length' of the erased line, and the line counter has to be incremented (one added) to take into accunt that there is now one less line in the display file.

Figure 2 shows the action of the scrolling routine that has left 2 lines and then scrolled once, deleting line 3.

Loading the Routine

The following method can be used to enter the routine into the ZX80. The routine is kept in a REM statement held off the screen so do not try to list it.

Enter the following lines:

10 REM 12345678901234567890123456 78901234567890123456789012345678901 2345678901234567890

20 REM *** SCROLL ***

30 REM FIRST POKE 16427 WITH THE NUMBER OF LINES TO BE LEFT THEN USE LET K=USR(16430)

Now push line 10 off the screen by using EDIT.

The actual steps are:

HOME
EDIT
RUBOUT,RUBOUT, ENTER 40 and NEWLINE.
EDIT
RUBOUT, RUBOUT, ENTER 70 and NEWLINE.
EDIT
RUBOUT, RUBOUT, ENTER 80 and NEWLINE.
EDIT
NEWLINE
LIST 20
and delete lines 40, 50, 60, 70 and 80.

The actual machine code can now be POKED into line 10 by using a simple loader.

The Assembly Language Listing

```
402B                             ORG 402B
402B 00           LEAVE          DEFB              No. of lines unaltered.
402C 00 00        NEW D-FILE     DEFW              Line end address store.
402E 2A 0C 40     START          LD  HL,(D-FILE)   Pick up D-FILE.
4031 ED 5B 0E 40                 LD  DE,(DF-EA)    Pick up DF-EA.
4035 22 2C 40     LINE END       LD  (NEW D-FILE),HL Address of scroll line.
4038 23                          INC HL            Enter next line.
4039 01 00 00                    LD  BC,+0000      Initialize counter.
403C 3E 75        NEXT           LD A,+75          Form end-of-line
403E 3C                          INC A             marker in A register.
403F EB                          EX  DE,HL         Change over registers.
4040 A7                          AND A             Clear carry flag.
4041 ED 52                       SBC HL,DE         Find if DF-EA has
4043 20 02                       JR  NC,NO ERROR   been reached.
4045 CF 09        ERROR          RST 0008, "9"     Will give "A" error.
4047 19           NO ERROR       ADD HL,DE         Reform HL.
4048 EB                          EX  DE,HL         Exchange back registers.
4049 BE                          CP  (HL)          Look for end of line.
404A 28 04                       JR  Z,COUNT       Yes. End of line found.
404C 23                          INC HL            No. So go to next
404D 0C                          INC C             character, incrementing
404E 18 EF                       JR  NEXT          counter and address.
4050 3A 2B 40 COUNT              LD  A,(LEAVE)     Collect the parameter.
4053 A7                          AND A             Is it zero?
4054 28 06                       JR  SCROLLING     Yes. So scroll.
4056 3D                          DEC A             No. So pass to next line.
4057 32 2B 40                    LD  (LEAVE),A     Replace the parameter.
405A 18 D9                       JR  LINE END      Back to LINE END.
405C 79           SCROLLING      LD  A,C           Save C in A register.
405D EB                          EX  DE,HL         Exchange registers.
405E A7                          AND A             Clear carry flag.
405F ED 52                       SBC HL,DE         Find length of rest of
4061 44                          LD  B,H           the display file and
4062 4D                          LD  C,L           put it in BC.
4063 2A 2C 40                    LD  HL,(NEW D-FILE) Collect the line end
4066 EB                          EX  DE,HL         address and scroll the
4067 ED B0                       LDIR              display file.
4069 2A 0E 40                    LD  HL,(DF-EA)    Reduce the value of
406C 4F                          LD  C,A           DF-EA by the size of
406D 37                          SCF               the character count
406E ED 42                       SBC HL,BC         of the scrolled line
4070 22 0E 40                    LD  (DF-EA),HL    that was saved in A.
4073 3A 25 40                    LD  A,(LINE COUNT) Increment the value
4076 3C                          INC A             of the system variable
4077 32 25 40                    LD  (LINE COUNT),A 16421- line count.
407A C9                          RET               Return to Basic.
```

```
100 FOR I=16427 to 16506
110 INPUT A
120 POKE I,A
130 NEXT I
```

The data for this routine is:

0,0,0,42,12,64,237,91,14,64,34,44,64,35, 1,0,0,62,117,60,235,167,237,82,32,2, 207,9,25,235,190,40,4,35,12,24,239,58, 43,64,167,40,6,61,50,43,64,24,217,121, 235,167,237,82,68,77,42,44,64,235,237, 176,42,14,64,79,55,237,66,34,14,64,58, 37,64,60,50,37,64,201

So enter lines 100 to 130 and RUN 100. Enter the machine code carefully. The checksum for the data is 6578, and this can be checked by adding the lines:

```
 90 LET T=0
110 delete
120 LET T=T+PEEK(I) and using RUN 90
140 PRINT T
```

Now delete all the lines from 90 onwards and SAVE.

Using the Scroll Routine

The following demonstration program shows in a simple way how the routine can be used.

With the routine stored in line 10, held off the screen, enter:

```
40 for I=1 TO 23
50 PRINT "LINE ";I
60 NEXT I
70 POKE 16427,2
80 LET K=USR(16430)
RUN
```

In line 70 always specify how many lines. Line 80 calls the scroll routine.

The result of the above program should be to produce a display in which "LINE 3" is missing, and the remainder of the display has been scrolled up a line (hence the gap between "LINE 23" and the error report).

ERROR A Report

The routine does declare an error when an attempt is made to hold more lines unscrolled than actually exist in the display file at that particular moment. This can be seen in the demonstration program by changing line 70 to read:

70 POKE 16427,24

This asks the routine to scroll all the lines after the 24th. Clearly a confusing situation so ERROR A is reported.

An Example Use of Scrolling

The following game shows just one of the many uses to which the scroll routine can be put. In this game you will test your skill at driving along a road. The scroll routine is used to scroll the "road" and also to remove an "end message"; in this case the message is "PRESS 5,6 or 8." Note carefully how the parameter of how many lines are to be left unscrolled is specified on each occasion that the scroll routine is called.

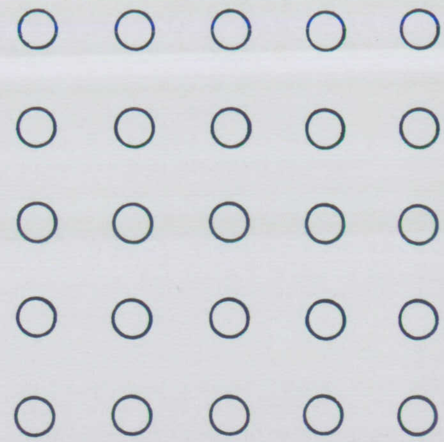
This game program is really only a first try at using the scroll routine, so I would therefore be very interested in seeing programs from readers who use this routine in writing their own programs.

Road Game

```
 10 as previously prepared and held off the screen.
 20 REM STOP                          , Just a reminder.
 30 RANDOMISE                         , Different every run.
 40 PRINT ,"*** ROAD GAME ***"        , Title.
 50 PRINT                             , Space.
 60 LET C=0                           , Set odometer to zero.
 70 GO TO 190                         , Go past subroutine.
 80 LET S=S+RND(5)-3                  , Move road a little left or right,
 90 IF S<4 THEN LET S=4               , but not off the left,
100 IF S>30 THEN LET S=30             , or off the right.
110 FOR A=1 TO S-3                    , Print "one mile" of road.
120 PRINT "█";   █ (shift A)          , Hard verge.
130 NEXT A                            ,
140 PRINT "    "; (4 sp.)             , Print four spaces for the road.
150 FOR A=S+2 TO 32                   , The other hard verge.
160 PRINT "█";   █ (shift A)          ,
170 NEXT A                            ,
180 RETURN                            , One mile of road printed.
190 FOR R=1 TO 4                      , Print four miles of road
200 LET S=15                          , but fixed, rather than curved.
210 GO SUB 110                        , Each mile.
220 NEXT R                            ,
230 FOR R=1 TO 3                      , Now print three miles of road
240 GO SUB 80                         , that does use the random function
250 NEXT R                            , to move the road left or right.
260 LET T=110+PEEK(16396)+PEEK(       , Initial car position.
16397)*256                            ,
270 LET R=T                           , Copy it.
280 POKE R,20                         , Print the car.
290 PRINT                             , Add an end of line marker.
300 PRINT                             , Space.
310 PRINT ,"PRESS 5,6 OR 8"           , Left, right, or straight ahead.
320 INPUT A                           , Collect direction.
330 POKE 16427,9                      , Hold the title and road unscrolled.
340 LET T=USR(16430)                  , Scroll away the blank line.
350 POKE 16427,9                      , Hold the title and road unscrolled.
360 LET T=USR(16430)                  , Scroll away 'PRESS 5,6 OR 8'
370 IF A=8 THEN LET R=R-(RND(5)       , Turn to right, but not always.
)2)
380 IF A=5 THEN LET R=R+(RND(5)       , Turn to left, but not always.
)2)
390 POKE 16427,2                      , Hold the title unscrolled.
400 LET T=USR(16430)                  , Scroll the road
410 GO SUB 80                         , Add another mile of road.
420 IF NOT PEEK(R)=0 THEN GO TO       , Test to see if runs off road.
450
430 LET C=C+1                         , Increment odometer.
440 GO TO 280                         , "Survived" so drive on.
450 POKE R,148                        , "CRASHED"
460 PRINT                             , Add end of line marker.
470 PRINT                             , Space.
480 PRINT "* YOU CRASHED AFTER        , The sign of failure.
```

puzzles & problems

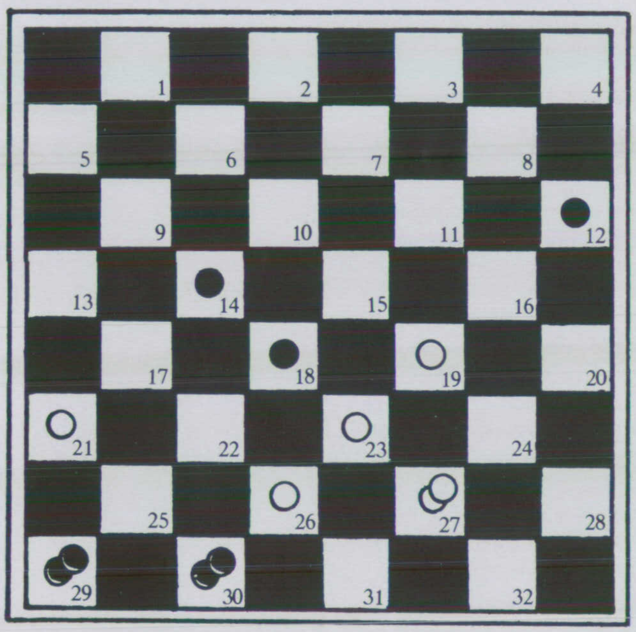
The Take-Away Game



Let's start off with a game you might care to program for your ZX80 computer. Lay out five rows of five coins each on the table. Each player, in turn, may remove one or more coins from any row or column of coins. However, there cannot be a gap between any of the coins. The coins removed must be contiguous within the row or column. To illustrate, suppose the first player removes coins number 3 and 4 from the top row of coins. His opponent could not then remove coins 1, 2 and 5 from this row because there would be a gap between coins 2 and 5. This player could, however, remove coins 1 and 2, or 5 from this row. The person who is forced to remove the last coin from the board is the *loser* of the game.

Find the Numbers

If we add the digits of a two-digit number together, we get the sum of 5. Now, if we write this two-digit number down, reverse it, and subtract the smaller number from the larger number, we find that the difference is 27. Can you tell us what these two numbers are?



Two-for-One

Here we have two checker puzzles. The first one is a checker problem and is pictured at the left. White has the move and should win in seven moves. Can you solve it? The second problem is straight forward enough. How many squares are there on a checkerboard? If your answer is 64, sit down and give someone else a chance. (From *Merlin's Puzzler 3* by Charles Barry Townsend, published by Hammond, Inc.)

An Easy Creditor

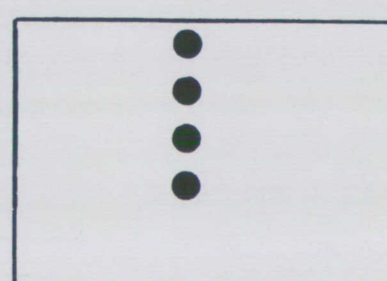
A gentleman was in temporary need of money. A friend lent him sixty dollars, telling him to repay it in such sums as might suit his convenience. Shortly afterwards he made a payment on account. His second payment was half as much as the first; his third three-quarters as much, his fourth one-quarter as much and his fifth two-fifths as much. It was then found, on striking a balance, that he still owed two dollars.

What was the amount of the first payment?

(From *Puzzles Old & New* by Professor Hoffmann circa 1890)

The Farmer and His Four Sons

Once upon a time (now where have I heard that before) a farmer owned a square field which had four apple trees growing on it. The trees were neatly spaced in a row as shown in the drawing at the right. The farmer had four sons that he wanted to divide the field among. His problem was that each son had to have an identically shaped piece of the field. Also, each piece had to be the same size in area. Finally, each piece of the field had to have one apple tree on it. If you had been the farmer, how would you have divided up the field so that each of the sons would receive his fair share?

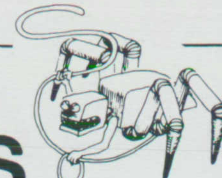


That's it for this issue. I hope you have enjoyed the problems brought by Merlin. If you have a favorite puzzle that you would like to share with the readers of *SYNC*, send it along. If Merlin uses it, he will send you a copy of *Merlin's Puzzler*, a great book filled with the best in puzzles and games.

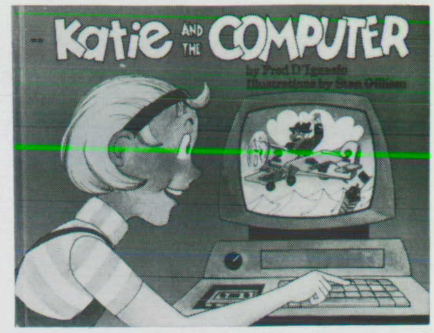
Until next time, keep puzzling!

Your editor,
Charles Barry Townsend

Answers on page 27.



new friends for your child...



Katie and the Computer

Fred D'Ignazio and Stan Gilliam have created a delightful picture book adventure that explains how a computer works to a child. Katie "falls" into the imaginary land of Cybernia inside her Daddy's home computer. Her journey parallels the path of a simple command through the stages of processing in a computer, thus explaining the fundamentals of computer operation to 4 to 10 year olds. Supplemental explanatory information on computers, bytes, hardware and software is contained in the front and back end papers.

Thrill with your chidren as they join the Flower Bytes on a bobsled race to the CPU. Share Katie's excitement as she encounters the multi-legged and mean Bug who lassoes her plane and spins her into a terrifying loop. Laugh at the madcap race she takes with the Flower Painters by bus to the CRT.

"Towards a higher goal, the book teaches the rewards of absorbing the carefully-written word and anticipating the next page with enthusiasm..."

The Leader

"Children might not suspect at first there's a method to all this madness—a lesson about how computers work. It does its job well."

The Charlotte Observer

"...the book is both entertaining and educational."

Infosystems

The book has received wide acclaim and rave reviews. A few comments are:

"Lively cartoon characters guide readers through the inner chamber of the computer."

School Library Journal

"...an imaginative and beautifully conceived children's story that introduces two characters—the Colonel and the Bug—who already seem to have been classic children's story book characters for generations."

The Chapel Hill Newspaper

Written by Fred D'Ignazio and illustrated in full color by Stan Gilliam. 42 pages, casebound, $6.95. (12A)

A t-shirt with the Program Bug is available in a deep purple design on a beige shirt. Adult size S, M, L, XL. Children's size S, M, L. $5.00.

To order, send a check for books plus $2.00 shipping and handling per order to Creative Computing, P.O. Box 789-M, Morristown, NJ 07960. NJ residents add 5% sales tax. Visa, MasterCard, and American Express orders are welcomed. For faster service, call in your bank card order toll free to **800-631-8112** (in NJ call 201-540-0445). Or use the handy order form bound into this magazine.

INVENTIVE PROGRAMS

FOR THE ZX80 4K ROM/1K RAM

Games, subroutines, and teaching aids in 4 K BASIC and MACHINE CODE. All listings include statement/subroutine logic and 8K ROM/16K RAM appendix.

PRICE: $1.00 per program listing
(minimum order 10, plus $2.50 S & H)

PARTIAL LIST * (4K/1K)

- ★ FLIP-A-COIN
- ★ 1 KEY BANDIT
- ★ TURRET GUNNER
- ★ ZX80 BASIC TEST
- ★ BLACKJACK
- ★ FIBONACCI SEQUENCE
- ★ COORD GRAPHICS
- ★ BARRAGE
- ★ ACEY-DUCEY II
- ★ LINE RENUMBER
- ★ TEST PATTERN
- ★ CASSINDEX
- ★ FOIL FENCING
- ★ BANG! FOR 1
- ★ EASTER DATE
- ★ STATE CAPITALS TEST
- ★ MACHINE CODE DISPLAY
- ★ GUILLOTINE
- ★ KEYBOARD GRAPHICS
- ★ 81 BOXES

TO ORDER

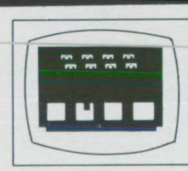
Specify programs wanted and enclose payment.

ZETA SOFTWARE
P.O. BOX 3522
GREENVILLE, S.C. 29608

* Complete list sent with your order, or send SASE for FREE list and order blank.

MACRONICS

ZX80-4K ROM quality software...

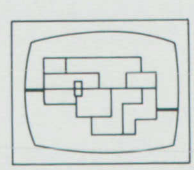


1K Space Intruders $20.00

Written in machine code to give fast moving 'Flicker Free' graphics. A superb achievement. Supplied as a listing only.

16K High Resolution $20.00 (192x192) graphic pictures

Never before achieved on the ZX80. Create your own artwork. Supplied as a listing (Ø5.00 extra if cassette required) with full screen demonstration picture.



9K Nightmare Park $15.00

A good example of a BASIC program using MACRONICS 'Amazing Active Display' The park of DEATH - can you get through this nightmare. No human as yet succeeded. Confronted by all sorts of 'Happenings' you'll be taken to your wits end. Supplied on cassette only.

Send International cheque or money order to:-
MACRONICS, 26 Spiers Close, Knowle, Solihull
West Midlands B93 9ES, United Kingdom

other MACRONICS software available from IMAGE COMPUTER PRODUCTS in the United States

Setting Up Bar Charts

Jon Passler

A bar chart is one of the most commonly used methods of graphically presenting data for quick interpretation. Such charts work nicely within the constraints of the ZX80 and MicroAce. Beside making for a good display, they provide an excellent way of storing data.

The program listed here works with 1K to chart two years of monthly checking account balances with vertical bars. The graph is set up for a range of $0 to $1500, but can be modified for other ranges with a few changes and some trial-and-error experimentation. Of course, any other sort of data such as monthly rainfall or average temperatures, miles-per-gallon, electricity use, or frequency distributions (histograms) can be plotted.

Because of memory limitations the array storing the data is created and filled in a routine that is later erased (lines 10-100). All elements of the array contain either data or zeroes, and line 320 is used to show the user which element of the array should be filled next. To add a monthly figure enter 330 LET B(16)=XXXX, then GO TO 330 and N/L, and finally erase line 330 and update line 320 to REM B(17).

After entering the program, you can enter the following data to see how it works: 1012, 796, 931, 1236, 1252, 1088, 786, 1132, 1194, 908, 1113, 896, 913, 849, 553, 429.

Jon Passler, 344 Cabot St., Beverly, MA 01915.

Subroutine

```
 10 DIM B(23)
 20 FOR I=0 TO 23
 30 LET B(I)=0
 40 NEXT I
 50 FOR I=0 TO 23
 60 PRINT I
 70 INPUT B(I)
 80 IF B(I)=0 THEN STOP
 90 IF I>20 THEN CLS
100 NEXT I
Erase lines 10-100 and use GO
TO 1 instead of RUN hereafter.
```

Bar Chart Program

```
100 REM GO TO 1   (Leave cursor on line
110 PRINT          100 when saving
120 PRINT          reminder not to RUN.)
130 PRINT "X100    AVE DAILY BALANCE"  (3 sp.)
140 PRINT "-----(3 sp. and 17 SHIFT G)"
150 PRINT
160 FOR I=-15 TO 0
170 LET K=(I/3)*10-I*10/3
180 IF K=0 AND I<-9 THEN PRINT -I;"-";
190 IF K=0 AND I>-10 THEN PRINT " ";-I;"-"; (1 sp.)
200 IF K>0 THEN PRINT " -";          (2 sp.)
210 FOR J=0 TO 23
220 IF B(J)=0 THEN GO TO 280
230 LET D=B(J)+I*100
240 IF D<-25 THEN PRINT " ";  (1 sp.)
250 IF D>-26 AND D<25 THEN PRINT CHR$(7);
260 IF D>24 THEN PRINT CHR$(130);
270 NEXT J
280 PRINT
290 NEXT I
300 PRINT "       1980    (SHIFT E)    1981" (7, 4, and 3 sp.)
310 STOP
320 REM B(16)
```

Bisection Iteration Square Root Program

Mike Goins

```
 10 PRINT "SQUARE ROOT OF X"
 20 PRINT "ENTER X   "; (3 spaces)
 30 INPUT X
 40 PRINT X
 50 LET L=0
 60 LET H=182
 70 LET T=(L+H)/2
 80 LET K=X/T
 90 IF K=T  THEN GOTO 160
100 IF H-L < 2  THEN GOTO TO 160
110 IF K < T THEN GOTO 140
120 LET L=T
130 GOTO 70
140 LET H=T
150 GOTO 70
160 PRINT "ROOT IS    ";T (3 spaces)
170 PRINT
180 STOP
```

This program operates by means of bisection iteration, which is basically just a variation of the old high-low game. The size limitation of the integer basic (variable size) limits the maximum root to 181.

Besides the mathematical value, this square root program is handy for use as a subroutine to represent the distance between two points (using the Pythagorean theorem) in some game programs in which one might try to guess the location of an object and when in error to find out by how great a margin.

Mike Goins, P.O. Box 3341, Bristol, IN 37620.

NOW AVAILABLE

keyboard conversions

- **Standard Computer Keyboard**
- **Type programs in half the time**
- **Minimize errors**
- **Wired keyboard hooks up in minutes**

Plans for keyboard conversion with reverse video $10.00

Keyboard with complete parts and plans $55.00

Wired keyboard, complete with plans $75.00

Mail for information:

L.J.H. Enterprises

P.O. Box 6273, Orange, CA 92667

For information or Visa or MasterCard orders call (714) 772-1595. Shipping charge for U.S.—$5.00.

SUPER INVASION ON YOUR ZX80!

SYNC magazine says Super Invasion is the "...best action game we have seen for the ZX80."

DOUBLE BREAKOUT

DOUBLE BREAKOUT challenges you to get through two barricades, using two ball angles. With seven levels of play, DOUBLE BREAKOUT is hard to beat. You'll be amazed at the superb graphics in this 1K game. $14.95

SUPER ZX80 INVASION

SUPER ZX80 INVASION is a flicker free, moving graphics game with three levels of play. SUPER INVASION challenges your skill as you shift your craft left and right and fire lasers at the invading space ships. Added bonus—each cassette contains a more sophisticated 2K version. $14.95

MOVING GRAPHICS

★ FITS 1K BASIC MACHINE.
★ TOTALLY FLICKER FREE.
★ ALL PROGRAMS ON CASSETTE.

SOFTSYNC, INC.

P.O. Box 480, Murray Hill Station, New York, NY 10156

☐ Please send me ___ copies of SUPER ZX80 INVADERS @ $14.95 ea.
☐ Please send me ___ copies of DOUBLE BREAKOUT @ $14.95 ea.
Plus $1.50 shipping and handling.
I enclose check/money order for ___
Name ___ Address ___ City ___ State ___ Zip ___

Multi-Dimensional Arrays for the ZX80

Jamie O'Connell

How many times have you sat down to convert a program for the ZX80, only to find that the first line was 10 DIM A(10,10)? Chances are that you gave up and turned the page. The next time you do not have to turn the page because it is possible to simulate dimensioned arrays on the Sinclair through the use of a simple algorithm.

Many versions of Basic define a two-dimensional matrix by the command DIM A(X,Y); where the X is the row subscript, and Y, the column. Any location on the matrix can be accessed by specifying values for X and Y. For example, LET A(3,4)=9 assigns the value 9 to the element located at row 3, column 4.

On the ZX80, we define a one-dimensional/vector array containing as many elements as we need and then use a simple formula to locate a given element. For example, if we want to initiate a 10 by 10 matrix, the instruction DIM A(99) sets up 100 locations and the formula A(X+Y*10) =9 assigns to the element at (X,Y) the value 9. In order to save space, the first element in the array simulates A(0,0) and the last, A(9,9). If we take the first element to be A(1,1) (as it is in most Basics), then we would use the general formula A((x-1)+(y-1)*X). These formulae result in column-order storage: all of column 1 is stored before 2. To simulate row-order storage, use the formula A((y-1)+(x-1)*Y) as in Figure 1.

ZX80 Actual Location A(L)	Simulated Column-order Location L=x+y*3	Simulated Column-order Location L=(x-1)+(y-1)*3	Simulated Row-order Location L=(y-1)+(x-1)*4
A(0)	A(0,0)	A(1,1)	A(1,1)
A(1)	A(1,0)	A(2,1)	A(1,2)
A(2)	A(2,0)	A(3,1)	A(1,3)
A(3)	A(0,1)	A(1,2)	A(1,4)
A(4)	A(1,1)	A(2,2)	A(2,1)
A(5)	A(2,1)	A(3,2)	A(2,2)
A(6)	A(0,2)	A(1,3)	A(2,3)
A(7)	A(1,2)	A(2,3)	A(2,4)
A(8)	A(2,2)	A(3,3)	A(3,1)
A(9)	A(0,3)	A(1,4)	A(3,2)
A(10)	A(1,3)	A(2,4)	A(3,3)
A(11)	A(2,3)	A(3,4)	A(3,4)

Figure 1. Simulated Locations for a 3 by 4 Array

Jamie O'Connell, Apt. 17 Cricket Brook, Dover, NH 03820.

In similar fashion, arrays of any number of dimensions can be accessed. The element A(X,Y,Z,...) is located at A((x-1)+(y-1)*X+(z-1)*X*Y+(...)*X*Y*Z+...). A simple comparison shows how the ZX80 can simulate three-dimensional Basic:

ZX80 Basic

```
10 DIM A(59)
20 LET X=3
30 LET Y=1
40 LET Z=2
50 LET A((X-1)+(Y-1)*3+(Z-1)*3*4)=9
•
•
•
```

3-dimensional Basic

```
10 DIM A(3,4,5)
20 LET X=3
30 LET Y=1
40 LET Z=2
50 LET A(X,Y,Z)=9
•
•
•
```

The best way to illustrate the use of dimensioned arrays is by a demonstration program. The one offered below is fun because the movement of the ship is essentially random. You can never know where it is until you blow it up or actually have it captured. Note that in a 10 by 10 matrix the array location is the same as the simulated location: A(37) is equivalent to A(3,7). This allows direct input of the coordinates desired. The display routine illustrates a fairly standard procedure for the printing of a matrix.

Capture

Capture is similar in some respects to many other matrix manipulation games; but, instead of trying to hit the enemy, you must surround and immobilize him. If you do succeed in hitting his location, you lose the game.

You have a total of fifteen mines which you use to block the enemy's progress. For a capture, his progress must be blocked in every direction. The display will show you where he was on the *previous* move. You can always place a mine at this previous location, as he has to move one space on each turn.

Lines 10-50 set up the enemy's initial location. Lines 80-195 output the display which shows: the matrix, the previous enemy location, the number of mines left, and the location of the mines as you place them. Lines 210-60 decide the enemy's new location, test for capture, and check the remaining number of mines. Lines 370-430 input your mine placement coordinates and test for a hit on the enemy.

To vary the number of mines, change line 60. Line 370 was keyed in using the following space saving technique: 370 INPUT (SHIFT 5) PRINT " (SHIFT 8) ROW-COLUMN". If you fail to use this, the program will print error code 4 when run—every byte counts! When entering the coordinates at line 370, enter them both before hitting NEWLINE. Happy hunting!

Capture Program Listing

```
 1Ø RANDOMIZE
 2Ø DIM A(99)
 3Ø LET X=RND(1Ø)-1
 4Ø LET Y=RND(1Ø)-1
 5Ø LET A(Y+X*1Ø)=148
 6Ø FOR M=-16 TO Ø
 7Ø PRINT
 8Ø PRINT "I WAS LAST AT..."
 9Ø PRINT
1ØØ PRINT " Ø123456789"
11Ø FOR I=Ø TO 9
12Ø PRINT I;
13Ø FOR J=Ø TO 9
14Ø PRINT CHR$(A(J+I*1Ø));
15Ø NEXT J
16Ø PRINT
17Ø NEXT I
18Ø PRINT
19Ø PRINT "MINES=";-M
195 PRINT
2ØØ LET A(Y+X*1Ø)=Ø
21Ø FOR T=1 TO 64
22Ø LET I=RND(3)-2
23Ø LET I=I+X
24Ø IF I 9 OR I Ø THEN GO TO 22Ø
25Ø LET J=RND(3)-2
26Ø LET J=J+Y
27Ø IF J 9 OR J Ø THEN GO TO 25Ø
28Ø IF I=X AND J=Y THEN GO TO 25Ø
29Ø IF NOT A(J+I*1Ø)=2Ø THEN GO TO 33Ø
3ØØ NEXT T
31Ø PRINT "OOPS-CAPTURED"
32Ø STOP
33Ø LET X=I
34Ø LET Y=J
35Ø LET A(Y+X*1Ø)=148
36Ø IF M=Ø THEN GO TO 44Ø
37Ø PRINT " INPUT ROW-COLUMN"
38Ø INPUT R
39Ø IF A(R)=148 THEN GO TO 46Ø
41Ø CLS
42Ø LET A(R)=2Ø
43Ø NEXT M
44Ø PRINT "OUT OF MINES"
45Ø STOP
46Ø PRINT "YOU BLEW ME UP"
```

Capture Sample Run

```
I WAS LAST AT...

  0 1 2 3 4 5 6 7 8 9
0
1                    ■
2
3
4
5
6
7
8
9

MINES=16

 INPUT ROW-COLUMN

17

I WAS LAST AT...

  0 1 2 3 4 5 6 7 8 9
0
1                *
2          *     *
3          * * * *
4            ■   *
5          * * * *
6
7
8
9

MINES=3

 INPUT ROW-COLUMN

45

I WAS LAST AT...

  0 1 2 3 4 5 6 7 8 9
0
1                *
2          *     *
3          * * * *
4            * ■ *
5          * * * *
6
7
8
9

MINES=2
OOPS-CAPTURED
```

TR$ and LET A$=A$+B$ on the ZX80

Harry Doakes

String handling on the ZX80 is reasonably good. The 4K Integer Basic lets the user print, input, and compare strings, and do specialized routines that will transform numbers into strings or characters.

There are also some large holes in its string handling abilities though such as limited string truncation and no concatenation. It is not hard to figure out why: Integer Basic takes up less than 3600 bytes of space, since the character generator, about 500 bytes long, is also in the 4K ROM. The only command for changing the size of a string is TL$ (which stands for *T*runcate (shorten) from the *L*eft of the *$*tring). PRINT TL$("FRED") produces "RED"; TL$ chops off the leftmost character of the string in parentheses whether it is a literal string (like "FRED") or a variable (A$, for example). With TL$ you can trim as much as you like from a string—but only one byte at a time, and only from the left side.

Sinclair's Integer Basic has no string concatenation commands at all. In other words, there is nothing like LET A$=A$+B$. You cannot lengthen a string.

Other small Basics—for example, Radio Shack's Level 1—allow fewer string variables and only INPUT and PRINT commands. The ZX80 looks good by comparison, but comparison cannot fill those string-handling holes.

This program can.

Enter the program in Figure 1. Line 10 should contain 52 zeroes.

```
  10 REM 000000000000000000000000
00000000000000000000000000000
  20 FOR A=1 TO 52
  30 PRINT A;" ";
  40 INPUT B
  50 POKE 16426+A,B
  60 PRINT B,
  70 NEXT A
  80 INPUT A
  90 IF A=0 THEN STOP
 100 INPUT B
 110 POKE 16426+A,B
 120 PRINT A;" ";B,
 130 GO TO 80
```

Figure 1.

Harry Doakes, P.O. Box 10860, Chicago, IL 60610.

Run the program and enter the following numbers in order. The numbers in parentheses are just entry numbers. Do not key them in.

(1)	175	(2)	235	(3)	30	(4)	10
(5)	197	(6)	225	(7)	57	(8)	235
(9)	249	(10)	235	(11)	227	(12)	43
(13)	43	(14)	43	(15)	40	(16)	9
(17)	227	(18)	209	(19)	213	(20)	227
(21)	35	(22)	3	(23)	3	(24)	24
(25)	21	(26)	60	(27)	237	(28)	185
(29)	35	(30)	235	(31)	33	(32)	0
(33)	0	(34)	35	(35)	12	(36)	32
(37)	252	(38)	4	(39)	32	(40)	249
(41)	227	(42)	193	(43)	213	(44)	227
(45)	35	(46)	35	(47)	237	(48)	176
(49)	235	(50)	227	(51)	249	(52)	201

When you have keyed in the 52nd value, *stop* and *proofread* the contents of your screen *very carefully*.

If you find a mistake, type the entry number, hit NEWLINE, then enter the correct value. The new version will appear at the end of the list of numbers. When you have corrected all mistakes, enter 0.

The program listing should now look something like Figure 2. Delete lines 20 through 130, and you are ready to key in your own program.

You now have two new string functions. The first will truncate a string from the *right side*; a TR$, if it existed, would do the same thing. To perform the equivalent of

LET G$=TR$(G$)

you write:

LET G$=G$
RANDOMISE USR(16428)

After the routine is complete, a PRINT G$ would show that the final character is gone.

The second function is even more useful; it is the equivalent of

LET L$=L$+M$

The appropriate program lines are:

LET L$=L$
LET M$=M$
RANDOMISE USR(16427)

At the end of the routine, M$ no longer exists, and L$ contains both strings.

Each function will work with any string variable, A$-Z$.

A few caveats:

You *must* perform the LET commands immediately before the USR line. LET creates a new entry at the end of the list of variables in RAM. By performing those LETs, you put the strings you are working on at the end of the variable list so the USR routine knows just where to go to find them.

You can substitute LET, PRINT, IF...THEN, and most other commands for RANDOMISE. Remember, though, if

```
  10 REM ▟ FOR 2▒? OR T FOR CONT
INUE FOR =FFFC▒=? THEN =7▄▄</W P
OKE █7 FOR 5  7£4 LOAD ▪4 CONTIN
UE =? THEN =77 POKE ▙FOR = CONTI
NUE ?
```

Figure 2.

you create a new variable with LET, you will have to LET A$=A$ etc. again before performing any more of these string functions.

Do not try to TR$ a string with nothing inside. You *can* do it if, for example, Q$=" "—but *not* if Q$="". If CODE (Q$)=1, do not use the TR$ routine.

These routines will not work with literals—only string variables. You must put a string into a variable before you can perform these operations on it.

If you want to use line 10 for your program, you can renumber the REM line using the EDIT function. It will work without change if it is still the first line of the program; if not, you will have to find its new location to call the routines with USR. The routines should never begin beyond 16639, or you will have to alter the machine language program.

Figure 3 lists the routine in assembly language. The procedure is relatively straightforward. Each routine loads HL with the E-LINE value (which points to the end of the variable list); finds the end of the variable it wants to work on; shifts the variable list down, one byte at a time, until it reaches the end of the list; then loads the new value of E-LINE into the proper location.

Parts of the routine may seem more complicated than necessary. The extra code is used to avoid instructions using values between 64 and 127 decimal (40h-7Fh). These values cause screen distortion and quickly crash the system if the ZX80 tries to display them as characters in a string or program line. A program that cannot be safely listed is too impractical for general use, so this one avoids those values. The routine begins, for example, not with LD HL, (400A), but with a longer instruction sequence that avoids the unlistable values.

At the beginning of the routine, certain things are taken for granted: BC=0, the Z flag is reset, and H=40h. The first two are *always* true when a routine is called with a USR command. The third is true for any routine between USR(16384) and USR(16639), since the ZX80 simply loads HL with the starting point and does a JP (HL).

Finally, notice that the end of the routine HL=SP. Thus, PRINT USR(16427) will not only add two strings together, but will also tell you where the top of available memory is. PRINT USR(16427)-PEEK (16400)-256*PEEK(16401) will return the amount free memory remaining—but be sure that you have the strings in place to be worked on, or you will scramble your variables and you may crash the system.

decimal	hex	assembler	
175	AF	XOR A	A=0, Z=1
235	EB	EX DE, HL	D=40h
30, 10	1E0A	LD E, 0Ah	E=0Ah
197	C5	PUSH BC	HL=0
225	E1	POP HL	
57	39	ADD HL, SP	HL=SP
235	EB	EX DE, HL	DE=SP, HL=400Ah
249	F9	LD SP, HL	SP=400Ah
235	EB	EX DE, HL	HL=SP
227	E3	EX (SP), HL	HL=(400Ah), (SP)=SP
43	2B	DEC HL	(HL)=80h
43	2B	DEC HL	(HL)=01h
43	2B	DEC HL	(HL)=character
40, 9	2809	JR Z, 09h	jump to $+$
227	E3	EX (SP), HL	DE=HL
209	D1	POP DE	(DE)=character
213	D5	PUSH DE	
227	E3	EX (SP), HL	
35	23	INC HL	(HL)=01h
3	03	INC BC	BC=0002h
3	03	INC BC	
24, 21	1815	JR 15h	jump to finish
60	3C	INC A	$+$: A=01h
237, 185	EDB9	CPDR	BC=displacement
35	23	INC HL	(HL)=01h
235	EB	EX DE, HL	(DE)=01h
33, 0, 0	210000	LD HL, 0000	HL=0
35	23	INC HL	HL=-BC
12	0C	INC C	
32, 252	20FC	JR NZ, FCh	
4	04	INC B	
32, 249	20F9	JR NZ, F9h	
227	E3	EX (SP), HL	
193	C1	POP BC	BC=HL=displacement
213	D5	PUSH DE	
227	E3	EX (SP), HL	HL=DE
35	23	INC HL	
35	23	INC HL	(HL)=character
237, 176	EDB0	LDIR	finish: shift until BC=0, DE=E-LINE
235	EB	EX DE, HL	
227	E3	EX (SP), HL	HL=SP, E-LINE=E-LINE
249	F9	LD SP, HL	SP=SP
201	C9	RET	

Figure 3.

puzzle answers

Find the Numbers: 41 and 14

Two-for-One: (A) White to move and win: 26-22, 18-25, 21-17, 14-21, 19-16, 12-26, 27-31. (B) There are 204 squares in a checkerboard. Some are single squares, some are made up of 4 squares, 9 squares, and so on.

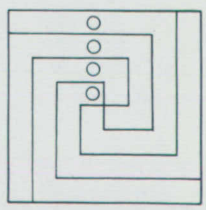
An Easy Creditor: The amount of the first payment was $20. To ascertain this amount, let x = the first payment. Then according to the conditions of the puzzle:

$$x + \frac{x}{2} + \frac{3x}{4} + \frac{x}{4} + \frac{2x}{5} + 2 = 60$$

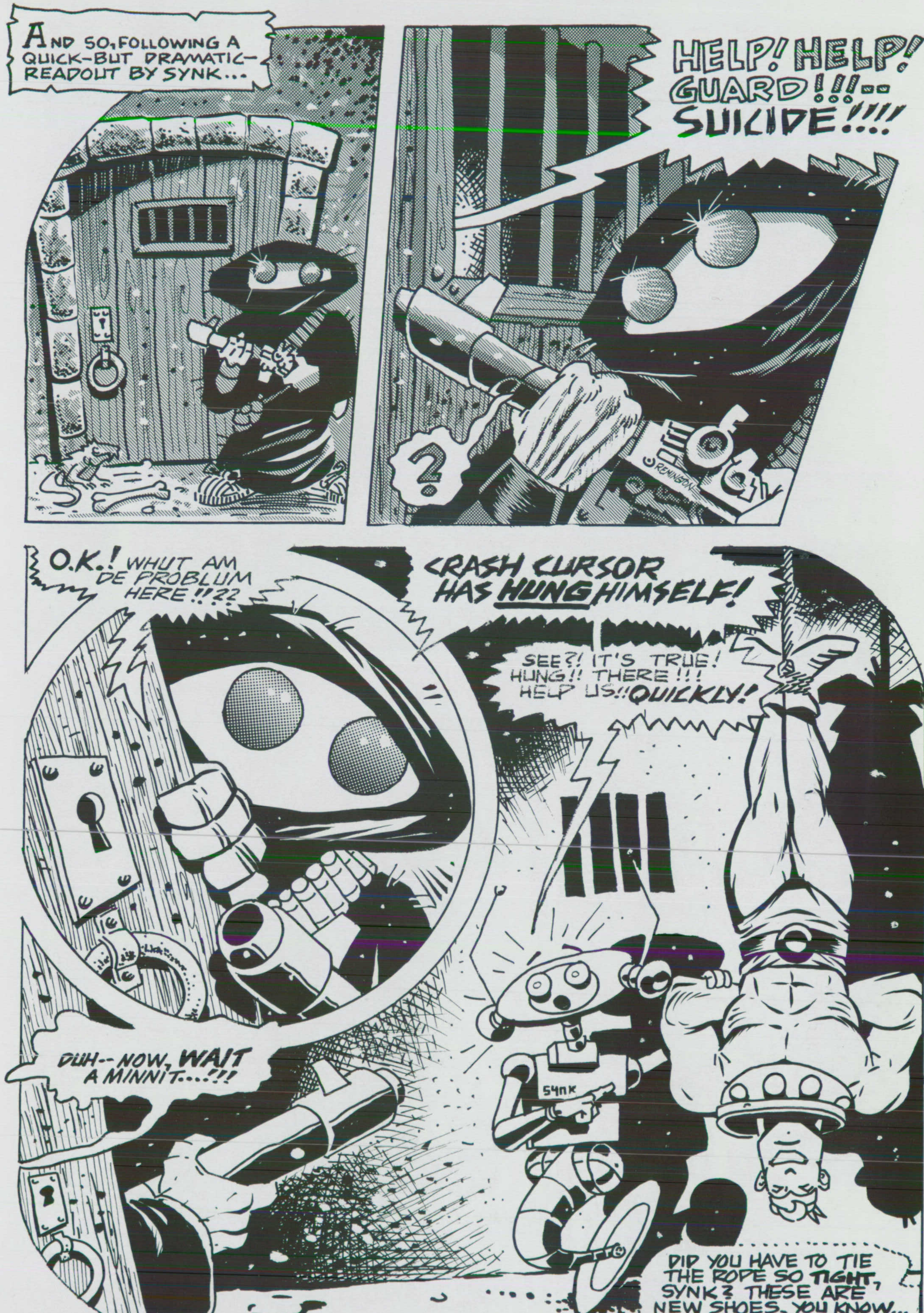
Multiplying by 20, the least common multiple of the various denominators,

$$20x + 10x + 15x + 5x + 8x + 40 = 1200$$
$$58x = 1200 - 40 = 1160$$
$$x = 20$$

The Farmer and His Four Sons:



CRASH CURSOR
4...
SO FAR: CRASH AND SYNK HAVE FALLEN INTO THE FEARFUL, NASTY, UNWASHED CLUTCHES OF DING THE MERCIFUL-NESSLESS! LEARNING OF DING'S PLOT TO SABOTAGE EARTH'S COMPUTER-SYSTEMS! THEN, DEEP IN DING'S DUNGEONS, GUARDED BY DING'S HATEFUL GLITCHOIDZ (TM), CRASH AND SYNK LEARN THAT THEIR TROUBLES HAVE JUST BEGUN...! ...FOR, ALSO A CAPTIVE OF THE MADMAN IS...
SNEER...
...KITTY! KITTY LITTER! MY TRUE HEART'S LOVE! MY ONE! MY ONLY! ER-- HOW WONDERFUL TO SEE YOU AGAIN!
STOW IT, CURSOR! WE WERE SUPPOSED TO GET MARRIED! REMEMBER?!
MARRIED? ER-- YES! OH YES!! BUT-- SOMETHING CAME UP! THE UNIVERSE NEEDED ME! GALACTIC CATASTROPHE...! AWFUL THING...
COOL THE SOLAR WIND, CRASH! YOU RAN OUT ON ME!
EVERYTHING WAS ALL READY!
...I'D BOUGHT A NEW WEDDING-GOWN AT GIMBLES!... A TUX FOR YOU (NOT EXPENSIVE, BUT NICE...)... FLOWERS! CATERING!... EVEN A HONEYMOON SUITE ON EXZEMA IV...!
SYNK
THERE HAS TO BE A WAY OUT OF HERE...
©1981 by TIMOTHY TRUMAN
1.
WAIT! I HAVE IT!!!
EXZEMA?!?
NO! A WAY OUT OF THIS JOINT!


AND SO, FOLLOWING A QUICK–BUT DRAMATIC–READOUT BY SYNK...
HELP! HELP! GUARD!!!.. SUICIDE!!!!
?
REMINGTON
O.K.! WHUT AM DE PROBLUM HERE!!??
CRASH CURSOR HAS HUNG HIMSELF!
SEE?! IT'S TRUE! HUNG!! THERE!!! HELP US!! QUICKLY!
DUH-- NOW, WAIT A MINNIT....!!!
SYNK
DID YOU HAVE TO TIE THE ROPE SO TIGHT, SYNK? THESE ARE NEW SHOES, YOU KNOW...


GEEZ! I ONLY TURNED MY BACK FOR A SECOND, TO CLEAN MY DISCS, AND WHEN I TURNED AROUND, THERE HE WAS-- TRUSSED UP LIKE AN ARCTURAN TURKEY!... HANGING THERE LIKE A MARTIAN HAM!!!
IT MUST HAVE BEEN THE SHOCK OF INCARCERATION! CRASH WAS A FREE SOUL, USED TO THE FREEDOM OF SPACE, ONE WITH THE GALAXY, THE WINDS OF SPACE BLOWING THROUGH HIS RECENTLY-WASHED HAIR! THE PULSE OF MALFUNCTIONING ROCKETS RAN THROUGH HIS BLOOD, HIS HEART KEEPING TIME WITH THE RUMBLE OF STAR-FIRE, HIS FEET WITH THE SOUND OF THE FOOTFALLS OF PURSUING CREDITORS!
AND NOW, IT'S COME TO THIS...!
WELL, DON'T JUST STAND THERE-- SAVE ME!
DUH-- YEAH! I GO! NOW! YUP I GET HELP! YUP...
SMASH!
SOON:
WELL-- HOW DO I LOOK?
WE'RE DOOMED !!!!
3
NEXT: CHAOS! COMPUTER-LASER BANDSAWS!! MORE...!!!

The ZX80 Makes the Grade

Lawrence Auer

Introduction

It is not a toy! Even with only 1K bytes of storage, the ZX80 can be an invaluable aid to the teacher in the calculation and evaluation of grades. In this article we present two programs running on the 1K Basic machine. The first determines the test scores and keeps track of which question caused the class the greatest difficulty. The second finds the class distribution of grades, enabling the teacher to scale the grades. While with more memory the two programs can be easily combined, presentation of the separate codes is made for those, like me, who want to do something while they wait for the 16K memory to arrive.

These programs have been used to handle the bi-weekly exams of twenty questions given to the 95 students in my introductory astronomy class. While, in principle, the same sort of computations could be accomplished at the university main frame computer, there is no comparison between that and a comfortable chair in front of the TV and ZX80. Further, because of the way the programs are set up, the data can be entered piecemeal, with more being added at your convenience. Finally, the tape storage system makes it a trivial task to keep the results for all the tests together. You can mount the cartridge and "instantly" see how the class did on any exam.

Program 1: Test Scoring

Using the ZX80 to add the scores on questions is an obvious and useful application. The ZX80 can be even more useful, however, because it can also keep track of which questions are being missed. After all the tests are graded, the teacher has a measure of which concepts were the hardest for the students. Knowing the question-by-question scoring, the teacher can see exactly which topics need review.

Before using the first program, edit lines 10 and 11. N is the number of questions on the test (20 or fewer; more may cause memory overflow). P is the default number of points per question. This number is used for the default when no number is explicitly entered in answer to the prompt at line 136. The reason for having a default P is that most answers are right (we hope!) and it is needless to have to enter numbers when just a single NEWLINE will do.

Having defined N and P appropriately, we can now enter the data. In response to the prompt "Q", where Q is the number of the question whose score is being requested, reply with the value earned. As described above, NEWLINE by itself will give the default value, P. Any other number is simply typed in the normal manner. The characters "K" and "E" are special and are interpreted as follows: 1) "K" stops (i.e., "kills") the program at this point. You can resume grading at exactly this point at a later time simply by using a CONTINUE command. More on this below. 2) "E" means the last score entered was in "error." The score is erased, and you are asked to correct its value.

After the scores on each question have been entered, the student's total is given. At this time you can get a plot of the relative number of points lost per question by typing "P" in answer to the prompt. The length of the bar is proportional to how many fewer points were earned by correct answers to this question than the one that earned the most points. Note that the *shortest bar* is the question the students did *best* on. The numeric value of the plotted quantity is listed at the end of the bar. After looking at the plot, you can either kill the program with "K" or go back to enter another exam by typing NEWLINE.

After stopping the program by entering "K" at any time, it may be SAVEd. All the relevant information is stored. The program and data may be LOADed, and you will start at the place where you stopped simply by executing CONTINUE.

Program 2: Grade Distribution

The overall performance of the class is determined by entering the test grades into Program 2. Grades are assumed to be in the range 0 to 100. A different range is easily accommodated by modification of the scaling used in line 225. Instead of 5, use another number which maps your input into the range (0,20). After a grade is entered, a plot of the number of students receiving a grade in each indicated interval is made. The first number in a row is the grade interval being plotted. The length of the bar labeled with the value G in the first column is proportional to the number of people who scored in the range, G to G+4; e.g., the bar labeled 75 contains all who scored 75 to 79. The second column contains the number who scored in this range; the bar length is proportional to this figure. The third is the cumulative distribution, i.e., how many were in this grade range or lower. The cumulative distribution is particularly important because it indicates the relative merit of a given absolute grade. The last value in this column is always just the number of test grades entered so far.

As in Program 1, the letter "E" entered instead of a grade deletes the grade just entered, i.e., it erases the error. If you type the letter "K", the program will stop. CONTINUE will start you again. SAVE preserves all data as well as the program, so you do not have to enter the grades again if something prevents you from finishing in one session.

Programming Suggestions of General Applicability

Several programming techniques used in these codes will be useful in other ZX80 programs.

When you have to enter the grades for 95*20=1900 answers, most of which are for full credit, you can get very tired of typing "5" then NEWLINE. It is much more efficient simply to type NEWLINE (NL in prompts) and let the machine make the default. Unfortunately, if you are INPUTing into a numeric field, NEWLINE by itself is ignored; thus, you have to use a string variable for INPUT. In this case, NEWLINE by itself sets the string to be the null string, "". The price paid is the

Lawrence Auer, 1301 Park Hills Ave., State College, PA 16801.

need to convert any non-null string to a numeric value. For example, one has to set G=15 when the input is the string "15", i.e., characters "1" and "5". The requisite code is in lines 148-151 of Program 1. Line 152 takes care of the default for null strings. Note the use of the operator " ", which permits the code to work even if A$ is initially null.

It is human (rather than machine!) to want to use mnemonic notation to enter signals for action. That is, when you want something erased, it is more natural to type "E" than to enter the value 101. When the input is being made into strings, it is no problem to check whether the string is the symbolic signal. This is the technique used in Program 1, line 265. In Program 2 the input is numeric, however. In order to use the letters "K" and "E" as symbolic signals there, we have to be a little more tricky. What we do in lines 3 and 4 of Program 2 is to define the variables E and K, giving them appropriate numerical values. The text typed in response to an INPUT statement is evaluated before being stored into the destination. Thus, typing E in response to the INPUT statement in line 200 is equivalent to entering the number 101. Typing the letter K there sets G=K=-1 and this value of G triggers the STOP command in line 201. As each of these variables can be set to any appropriate "impossible" test scores, it is no problem to make modifications if grades 100 are allowed. Finally, if you prefer, more dramatic and memorable variable names like KILL or ERASE can be used.

The CONTINUE Command

Of all the pieces of hidden gold in the ZX80, the most valuable is the command CONTINUE. When you leave the program execution mode either voluntarily because of a STOP statement in your program or involuntarily because of an error (like trying to write too much on the screen), the place you were when you stopped is remembered. While in immediate mode, you can do what you want, including resetting variables and even editing the program. The CONTINUE statement will start you again at the line following the one where you stopped. Thus, in these programs you can kill (i.e., "K") whenever you want, then SAVE the file on tape. The LOAD operation brings in the file with the program, all the data, and even the information on where you stopped. CONTINE following the LOAD acts just as if there had been no tape storage in between. That is, SAVE followed by LOAD completely re-establishes the environment as it was before the SAVE operation. The only thing to remember if you are going to use the STOP-CONTINUE trick is to make sure that the statement following the STOP produces a recognizable cue. If you just have the sequence, 100 STOP, 110 INPUT X, when you restart with CONTINUE you will have a nice blank screen and the cursor waiting for input with no hint as to what input is wanted. To avoid this problem in Program 1, for example, line 139 jumps back to the input prompt, so you can tell what is expected when you restart.

One of the problems in having only 1K of memory is that peculiar errors can occur when you are OUTPUTing. The characters to be written on the screen occupy the same memory as your program. You can, therefore, be running quite nicely, entering question scores, and then have trouble when you try the plot, because you do not have enough memory for the output. The choices made for the scaling of the bar lengths in these programs work well for my exams with 20 questions and 95 students, but there may be something unusual about your distributions. If you do have trouble, do not panic! Simply change the scaling; reduce the 15 in line 226 of Program 1 and/or the 20 in line 120 of Program 2. None of the data will be lost as long as you use CONTINUE to restart.

More memory will permit these programs to be significantly improved. Programs 1 and 2 could then be combined so that all you would ever enter would be the question scores. Also, one could add a subroutine which would give the test grade above which a specified fraction of the class scored. Finally, with 16K you should be able to store the names with the grades and thus have your "mark book" on tape. In any case, I hope that you find these programs as useful as I have even in their limited form.

Program 1: Test Scoring

```
10 LET N = 20
11 LET P = 5
20 DIM S(N)
25 LET T = 0
100 CLS
105 PRINT "SCORES?"
110 LET T = T + 1
115 LET R = 9999
116 LET Q = 0
120 LET Z = 0
130 FOR I = 1 TO N
135 PRINT "Q";I;
136 INPUT A$
137 IF NOT A$="K" THEN GOTO 140
138 STOP
139 GOTO 135
140 IF NOT A$="E" THEN GOTO 148
141 CLS
142 LET I = I - 1
143 LET Z = Z - G
144 LET S(I) = S(I) - G
145 GOTO 135
148 LET G = 0
149 LET G = 10*G + CODE(A$) - 28
150 LET A$ = TL$(A$)
151 IF A$ > "" THEN GOTO 149
152 IF G < 0 THEN LET G = P
153 PRINT G
155 LET Z = Z + G
160 LET S(I) = S(I) + G
165 IF S(I) < R THEN LET R = S(I)
166 IF S(I) > Q THEN LET Q = S(I)
170 NEXT I
171 LET R = Q - R
175 PRINT "P TO PLOT",
200 PRINT "GRADE",Z
205 INPUT A$
210 IF NOT A$ = "P" THEN GOTO 100
215 CLS
216 PRINT "REL ERRORS"
```

```
220 FOR I = 1 TO N
222 IF I < 10 THEN PRINT " ";
224 PRINT I;" ";
226 LET Z = 15*(Q - S(I))/R
227 IF Z = 0 THEN GOTO 233
230 FOR J = 1 TO Z
231 PRINT CHR$(128);
232 NEXT J
233 PRINT "■";Q-S(I)
250 NEXT I
255 PRINT "K TO KILL"
260 INPUT A$
265 IF A$ = "K" THEN STOP
270 GOTO 100
```

Program 2: Grade Distribution

```
  1 REM GRADE HISTOGRAM
  3 LET K = -1
  4 LET E = 101
 10 DIM C(20)
 15 LET A = 0
 20 LET N = 0
 25 LET S = 1
100 LET G = 0
110 FOR I = 0 TO 20
115 LET G = G + C(I)
120 LET L = 20*C(I)/S
122 IF I < 2 THEN PRINT " ";
123 PRINT 5*I
124 IF C(I) < 10 AND I < 20 THEN PRINT "■";
125 PRINT C(I);"■";G;
130 IF L = 0 THEN GOTO 150
135 FOR J = 1 TO L
140 PRINT CHR$(128)
145 NEXT J
150 PRINT "■"
155 NEXT I
160 IF N > 0 THEN PRINT "AV=";A/N;
198 REM K=KILL(STOP), E=ERROR
199 PRINT " GIVE GRADE, K OR E"
200 INPUT G
201 IF G > K THEN GOTO 205
203 STOP
204 GOTO 100
205 IF G < E THEN GOTO 215
206 LET D = -1
207 LET G = Q
208 GOTO 220
215 LET D = 1
216 LET Q = G
220 LET N = N + D
221 LET A = A + D*G
225 LET G = G/5
230 LET C(G) = C(G) + D
235 IF  C(G) > S THEN LET S = C(G)
240 CLS
245 GOTO 100
```


Blank Cassettes

The quality of cassette tape used to save and load programs is an important factor in getting the programs to run. Tape quality for computers is measured differently from quality for audio tape. The tape must be capable of sending to the computer the electronic signals of the program without transmitting extraneous noises that could interfere with the ability of the computer to load the tape.

Our blank cassettes are tested and recommended for computer use. C-10 cassette, 5 min. per side, blank label on each side in a Norelco hard plastic box. [0010] $1.25 each.

Head Cleaner

After hours of use, the read/write head in a cassette recorder will pick up minute particles of tape oxide. This dirt will hardly be noticable in dictation or music. But it is very noticable in computer use. One dropped bit in 16,000, and the program won't load.

Help keep your recorder in top shape with our non-abrasive head cleaner. It consists of 18 inches of stiff cleansing fabric in a standard cassette shell. One 10-second pass every 40 hours of use will keep your heads as good as new. [0011] $2.00. Send payment plus $1.00 Shipping per order to:

Peripherals Plus

39 East Hanover Avenue
Morris Plains, NJ 07950

MUSIC! for 4K ROM, 1K or more RAM. 2 octaves, 127 note length, any tempo. Songs repeat. Random sounds also. Cassette and insts. $6.95 pp. $10. outside U.S. **Wm Don Maples, 688 Moore St., Lakewood, CO 80215.**

Find ZX-80 Owners

Advertise in SYNC, the magazine exclusively dedicated to the Sinclair ZX-80 and The MicroAce. Call or write for details and a rate card. Let SYNC readers know who you are.

SYNC
39 E. Hanover Ave.
Morris Plains, New Jersey 07950

Multiplication Three-in-a-Row

Austin R. Brown, Jr.

"Multiplication Three-in-a Row" is based on the program "Multiplication Bingo," by Jean Wilson, Special Education teacher at Leadville High School, Leadville, Colorado. She was seeking a way to motivate students who were having difficulty learning to multiply and found that completing five in a row on a Bingo board helped supply the motivation. An array 5 by 5 is too big for the 1K ZX80, but 3 by 3 will fit.

The game proceeds as follows. You select a square on the board. You are then given a multiplication problem to solve. If you solve it within two tries, an "X" goes in the square. If you fail, an "O" goes in the square. If you get three X's in a row before the board is filled, you win. See Sample Run 1.

The program can be used to build skills in mental arithmetic, pencil and paper arithmetic, or calculator arithmetic. It can generate other ranges of problems by changing lines 120-130. For example, use RND(19) rather than RND(9) to generate factors into the teens.

This is not a tic-tac-toe game. The number or location of 0's does not matter, as long as you can get three in a row by the time the board is filled.

Programming Notes

The program is built upon the array U(N), N=1,...9, where N represents one of the squares on the board. If the square has not yet been used (the number of the square still shows in the display), U(N)=0. If the player has successfully solved a problem at that square, "X" shows in the display, and U(N)=1. If the player has failed to solve a problem at that square, "0" shows in the display, and U(N)=2.

Austin R. Brown, Jr., 407 Peery Parkway, Golden, CO 80401.

Use of the array helps the program logic in several ways, as shown in Listing 1. First, we can generate and update the display without the need for nine different string variables and the repetitious logic they require (lines 740-890). Second, we can easily check for an already occupied square,, since U(N) will no longer be zero (line 110). Third, we can also check for three-in-a-row, since we have success if, and only if, all three U's, and hence their product, are equal to one (lines 230-380). Fourth, we can easily tell when the game is over. As long as there is at least one unoccupied square, its U is zero (lines 300-320). Fifth, we record a right or wrong answer simply by changing the current U (lines 190 and 510).

Tic-Tac-Toe

The program can easily be adapted to a tic-tac-toe game either for two players or for one player against the computer. Success for "O" is tested as well as success for "X" in lines 230-280, except that the product must be eight.

Listing 2 shows the program modified for a two-person game, with the computer simply keeping track of the action. Sample Run 2 shows a game. Modifying the program to play computer against human is left as an exercise for the reader. For example, a simple strategy of random moves by the computer could be implemented by the following changes:

```
74 IF J=2 THEN GO TO 500
500 LET N=RND(9)
510 IF NOT U(N)=0 THEN GO TO 500
520 GO TO 120
```

This strategy can be bewildering to the human encountering it for the first time.

Notes:

REMarks should not be entered into the ZX80. They are included strictly to show the program logic.

Three-in-a-Row

```
  5 REM Multiplication 3-in-a-row
  6 REM A.R.Brown,Jr. 6/1/81
  7 REM Initialize
 10 RANDOMIZE
 20 DIM U(9)
 30 FOR I=1 TO 9
 40 LET U(I)=0
 50 NEXT I
 60 GO SUB 700
 75 REM Pick square
 80 PRINT "WHICH SQUARE?"
 90 INPUT N
100 GO SUB 700
110 IF N<1 OR N>9 OR NOT U(N)=0
THEN GO TO 80
115 REM Generate problem
120 LET A=RND(9)
130 LET B=RND(9)
140 PRINT "SQUARE ";N
160 PRINT "WHAT IS ";A;"*";B;"?
"
170 INPUT C
175 REM Check for correct answer
180 IF NOT C=A*B THEN GO TO 500
185 REM Right answer
190 LET U(N)=1
200 GO SUB 700
210 PRINT "RIGHT"
220 PRINT
225 REM Check for 3 in a row
230 FOR K=1 TO 3
240 IF U(K)*U(K+3)*U(K+6)=1 THE
N GO TO 900
250 IF U(3*K-2)*U(3*K-1)*U(3*K)
=1 THEN GO TO 900
260 NEXT K
270 IF U(1)*U(5)*U(9)=1 THEN GO
TO 900
280 IF U(3)*U(5)*U(7)=1 THEN GO
TO 900
295 REM Check for end of game
300 FOR I=1 TO 9
310 IF U(I)=0 THEN GO TO 80
320 NEXT I
355 REM Losing end
360 PRINT "SORRY, YOU LOSE"
370 GO TO 999
495 REM Check error
500 IF F>0 THEN GO TO 600
505 REM 2nd time, answer & move
510 LET U(N)=2
520 GO SUB 700
530 PRINT A;"*";B;"=";A*B
540 GO TO 300
595 REM 1st time, try again
600 LET F=-1
610 PRINT "WRONG,"
620 GO TO 160
695 REM Output tableau
700 CLS
710 LET F=1
720 PRINT "MULTIPLY 3-IN-A-ROW"
730 PRINT
740 PRINT " ";
750 FOR I=1 TO 9
760 LET A$=STR$(I)
770 IF U(I)=1 THEN LET A$="X"
780 IF U(I)=2 THEN LET A$="O"
790 PRINT A$;
800 IF NOT I=3*(I/3) THEN PRINT
" █ "; (Shift A; 1 time)
810 IF I=3 OR I=6 THEN GO SUB 8
60
820 NEXT I
830 PRINT
840 PRINT
850 RETURN
860 PRINT
870 PRINT "███████████"
                    (Shift A; 11 times)
880 PRINT " ";
890 RETURN
895 REM Winning end
900 PRINT "-- HOORAY, YOU WIN -
-"
```

Listing 1.

Tic-Tac-Toe

```
  5 REM Tic-Tac-Toe
  6 REM A.R.Brown,Jr. 6/6/81
  7 REM Initialize
 10 RANDOMIZE
 20 DIM U(9)
 30 FOR I=1 TO 9
 40 LET U(I)=0
 50 NEXT I
 60 GO SUB 700
 65 LET J=0
 69 REM Pick square
 70 LET J=J-(J/2)*2+1
 72 IF J=1 THEN PRINT "X MOVES
"
 74 IF J=2 THEN PRINT "O MOVES
"
 80 PRINT "WHICH SQUARE?"
 90 INPUT N
100 GO SUB 700
110 IF N<1 OR N>9 OR NOT U(N)=0
THEN GO TO 80
120 LET U(N)=J
130 GO SUB 700
225 REM Check for 3 in a row
230 FOR K=1 TO 3
235 LET I=U(K)*U(K+3)*U(K+6)
240 IF I=1 OR I=8 THEN GO TO 90
0
245 LET I=U(3*K-2)*U(3*K-1)*U(3
*K)
250 IF I=1 OR I=8 THEN GO TO 90
0
260 NEXT K
265 LET I=U(1)*U(5)*U(9)
270 IF I=1 OR I=8 THEN GO TO 90
0
275 LET I=U(3)*U(5)*U(7)
280 IF I=1 OR I=8 THEN GO TO 90
0
295 REM Check for end of game
300 FOR I=1 TO 9
310 IF U(I)=0 THEN GO TO 70
320 NEXT I
360 PRINT " - T I E -"
370 GO TO 999
695 REM Output Tableau
700 CLS
710 LET F=1
720 PRINT "TIC-TAC-TOE"
730 PRINT
740 PRINT " ";
750 FOR I=1 TO 9
760 LET A$=STR$(I)
770 IF U(I)=1 THEN LET A$="X"
780 IF U(I)=2 THEN LET A$="O"
790 PRINT A$;
800 IF NOT I=3*(I/3) THEN PRINT
" █ ";      (Shift A; 1 time)
810 IF I=3 OR I=6 THEN GO SUB 8
60
820 NEXT I
830 PRINT
840 PRINT
850 RETURN
860 PRINT
870 PRINT "███████████"(Shift A; 11 times)
880 PRINT " ";
890 RETURN
895 REM Winning Ends
900 IF I=1 THEN PRINT "X WINS "
910 IF I=8 THEN PRINT "O WINS "
```

Listing 2.

Sample Run 1

```
MULTIPLY 3-IN-A-ROW

 1 █ 2 █ 3
███████████
 4 █ 5 █ 6
███████████
 7 █ 8 █ 9

WHICH SQUARE?

SQUARE 1
WHAT IS 9*8?
```

We gave 72 for the answer

```
MULTIPLY 3-IN-A-ROW

 X █ 2 █ 3
███████████
 4 █ 5 █ 6
███████████
 7 █ 8 █ 9

RIGHT

WHICH SQUARE?

SQUARE 5
WHAT IS 6*9?
WRONG,
WHAT IS 6*9?
```

We gave first 69, then 54 for the answer

```
MULTIPLY 3-IN-A-ROW

 X █ 2 █ 3
███████████
 4 █ X █ 6
███████████
 7 █ 8 █ 9

RIGHT

WHICH SQUARE?

SQUARE 9
WHAT IS 7*4?
WRONG,
WHAT IS 7*4?
```

We gave first 21, then 14 for the answer

```
MULTIPLY 3-IN-A-ROW

 X █ 2 █ 3
███████████
 4 █ X █ 6
███████████
 7 █ 8 █ O
```

Sample Run 2

```
TIC-TAC-TOE

 1 █ 2 █ 3
███████████
 4 █ 5 █ 6
███████████
 7 █ 8 █ 9

X MOVES,
WHICH SQUARE?
```

X chooses 5.

```
TIC-TAC-TOE

 1 █ 2 █ 3
███████████
 4 █ X █ 6
███████████
 7 █ 8 █ 9

O MOVES,
WHICH SQUARE?
```

O chooses 8.

```
TIC-TAC-TOE

 1 █ 2 █ 3
███████████
 4 █ X █ 6
███████████
 7 █ O █ 9

X MOVES,
WHICH SQUARE?
```

X chooses 7.

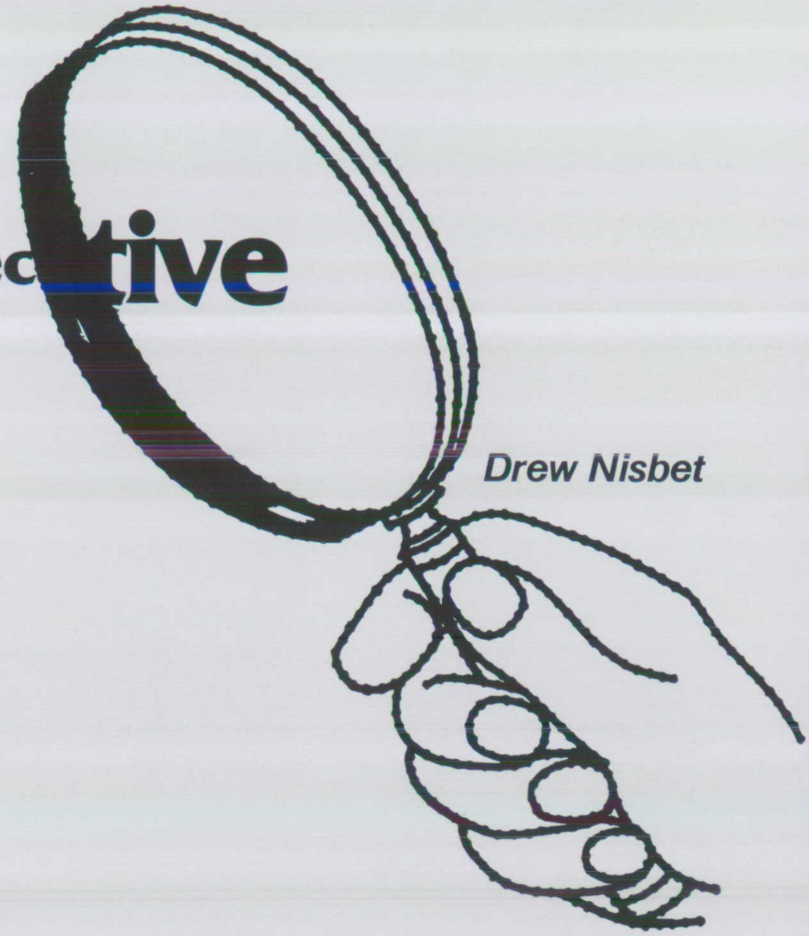
```
TIC-TAC-TOE

 1 █ 2 █ 3
███████████
 4 █ X █ 6
███████████
 X █ O █ 9

O MOVES,
WHICH SQUARE?
```

O chooses 3.

Detective



Drew Nisbet

A murder has been committed and the perpetrator has threatened to strike again! It is up to you to uncover the two pieces of evidence which will identify the murderer before he can carry out his threat.

The game consists of searching the 4 rooms in the building where the crime occurred for the incriminating weapon and fingerprints. Your initial location is randomly selected as are the locations of the gun and the prints. The amount of time allocated to you ranges from 6 to 30 minutes. To remain in your current position or to move in either a clockwise or counter clockwise direction requires from 1 to 5 minutes. A diagonal move can take from 2 to 9 minutes.

To search for one piece of evidence requires from 1 to 5 minutes; to search for both requires from 2 to 9 minutes. If your allotted time drops below 6 minutes one of your associates may search a room for you and declare it "clean" and therefore you do not have to search it yourself, although you already may have done so.

If you run out of time, the locations of the fingerprints and the gun are displayed. If you locate the evidence, the amount of time remaining is printed.

The program is loaded in three sections. First, an array of 72 print characters is set up. This array contains the floor plan display.

```
1000 DIM A(71)
1010 FOR I = 0 TO 71
1020 IF (I/12) * 12 = I THEN CLS
1030 PRINT I + 1,
1040 INPUT K
1050 LET A(I) = K
1060 PRINT A(I)
1070 NEXT I
```

Run this portion of the program and input the 72 character codes listed below. If you make an error in entering the values, you can either rerun the routine or correct individual entries with a LET statement (e.g., LET A(O) = 135).

```
135,131,131,131,131,131,131,131,131,131,131,134

  2,  0, 30,  0,  0,  0,  0,  0,  0, 31,  0,130

133,  3,  0,  3,132,  0,  0,133,  3,  0,  3,132

  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,130

  2,  0, 29,  0,  0,  0,  0,  0,  0, 32,  0,130

133,  3,  3,  3,132,  0,  0,133,  3,  3,  3,132
```

After the first portion has been run, delete lines 1010 to 1070 inclusive and enter the following lines:

```
170 LET K = 0
180 FOR I = 1 TO 6
190 FOR J = 1 TO 12
200 PRINT (CHR$(A(K));
210 LET K = K + 1
220 NEXT J
230 PRINT
240 NEXT I
250 STOP
```

Drew Nisbet, 6 Moffatt Crt., Toronto, Ont., Canada. M9V4E1.

Key GO TO 170 and NEWLINE and check the display. If it requires correction use a LET statement as above. If the display is functioning properly, it would be a good idea to save the partial program at this point.

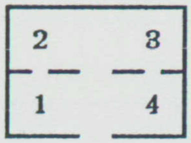
Now enter the main body of the program:

Delete line 1000 and save the program. To execute, key GO TO 100 rather than RUN as the latter will clear the print codes stored in array "A". This program could easily be altered in order to create other "Search and Find" games. By changing the names of the articles to be searched for and by setting up an appropriate display for the top of the screen this program could be used as a basis for a "treasure hunt," "spy" or similar game where it is necessary to locate something that is hidden.

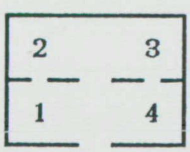
```
100 RANDOMISE
110 LET G = RND(4)                       LOCATION OF GUN
120 LET P = RND(4)                       LOCATION OF PRINTS
130 LET R = RND(4)                       STARTING ROOM
140 LET M = RND(25) + 5                  AMOUNT OF TIME
150 LET GF = 0                           GUN FOUND SWITCH
160 LET PF = 0                           PRINTS FOUND SWITCH
250 PRINT "ROOM=";R,"TIME=";M
260 LET Q = RND(5)                      TIME FOR CIRCULAR MOVE
270 PRINT "SEARCH?"
280 INPUT Y$
290 IF Y$ = "N" THEN GO TO 470
300 PRINT "1-GUN"
302 PRINT "2-PRINTS"
304 PRINT "3-BOTH"
310 INPUT F
320 LET M = M - Q
330 IF F = 3 THEN LET M = M - G
340 IF M < 0 THEN GO TO 600
350 IF F = 2 THEN GO TO 430
380 IF NOT G = R THEN GO TO 420      CHECK FOR GUN
390 LET GF = -1
400 PRINT "GUN FOUND"
420 IF F = 1 THEN GO TO 460
430 IF NOT P = R THEN GO TO 460      CHECK FOR PRINTS
440 LET PF = -1
450 PRINT "PRINTS FOUND"
460 IF GF AND PF THEN GO TO 700
470 LET T = RND(4)
480 IF M < 6 AND NOT (T = G OR T = P OR T = R) THEN PRINT
T;" CLEAN"
490 PRINT "ROOM?"
500 INPUT S
510 IF S < 1 OR S > 4 THEN GO TO 500
520 CLS
530 LET M = M - Q
540 IF ABS(S - R) = 2 THEN LET M = M - G
550 IF M < 0 THEN GO TO 600
560 LET R = S
570 GO TO 170
600 PRINT "OUT OF TIME"
610 PRINT "G:";G,"P:";P
620 STOP
700 PRINT "TIME=";M
```

To run, key GOTO 100. *Do not* use RUN.

Sample Run




```
ROOM=4    TIME=24
SEARCH?
( 'Y' - N/L)
1-GUN
2-PRINTS
3-BOTH
( '3' - N/L)
GUN FOUND
ROOM?
( '1' - N/L)
.
.
.
ROOM=2    TIME=5
SEARCH?
( 'Y' - N/L)
1-GUN
2-PRINTS
3-BOTH
( '2' - N/L)
1 CLEAN
ROOM?
( '3' - N/L)
OUT OF TIME
G:4    P:3
```




```
ROOM=2    TIME=16
SEARCH?
( 'Y' - N/L)
1-GUN
2-PRINTS
3-BOTH
( '3' - N/L)
GUN FOUND
PRINTS FOUND
TIME=8
```

A Parallel Interface for the ZX-80/MicroAce Computer

Alger Salt

Introduction

Almost everyone who owns a computer will ask or be asked, "What sort of practical things can it do?" One of the most obvious practical applications is controlling external devices; however, few microprocessors or CPUs are designed to do this directly.

Most manufacturers of microprocessors offer devices called peripheral controllers which are integrated circuits designed to be compatible with their particular CPU. These controllers greatly simplify the task of interfacing external peripheral devices such as disk drives, terminals, and printers. Fortunately, the engineers at Sinclair Research Limited chose to design their microcomputer around the Z-80 CPU which is well-supported by several excellent peripheral controllers. One of these, the Z-80 PIO can be used in constructing a simple parallel interface for the ZX80/MicroAce computers.

Overview of the Z-80 PIO

The Z-80 PIO is a 40 pin integrated circuit designed to serve as a simple direct, TTL compatible interface between the Z-80 CPU and peripheral devices employing parallel data transfer. (See Figure 1.) Communication between the PIO and the CPU is accomplished by connecting the PIO data lines directly to the CPU data bus. The PIO is a two-port device. This means it can send and/or receive two sets of 8-bit parallel data. Control lines on the PIO select one of the two ports (B/$\overline{A}$ SEL), enable the PIO ($\overline{CE}$), and allow the PIO to differentiate control words from data words (C/$\overline{D}$ SEL). Three other control lines ($\overline{M1}$, $\overline{IORQ}$, $\overline{RD}$) insure proper timing sequences during CPU I/O operations. The bars over the signal names indicate that they are active low.

Each port has two control lines used to establish handshaking between the PIO and the peripheral device. These two control lines (RDY and $\overline{STB}$) are sometimes, though not always, necessary to synchronize data transfer. In other words, one of these control lines, the RDY line, may be activated to tell a device which is sending data to the PIO, "Do not send data now. I am not ready... O.K., now I am ready. Send data." The device may respond by activating the STB line, "O.K., here is the data. Get it now so I can do something else." By using the handshake lines, communication is established between the PIO and the peripheral device resulting in an orderly, efficient transfer of data.

Alger Salt, East Carolina University, Chemistry Department, Greenville, NC 27834.

The PIO contains a number of internal registers used to control its operation. The most important is the 2-bit mode control register which can be programmed to select one of several operating modes on port A or port B.

The PIO may be operated in one of four modes, designated mode 0 through 3. Mode 0 is the output mode; all eight lines on the designated port are output to a device. In mode 1, the input mode, all lines on the port are input from a device. Mode 2 is the bidirectional mode and is restricted to port A. In this mode the handshake lines of port B along with the port A handshake lines are used to control the flow of data in both

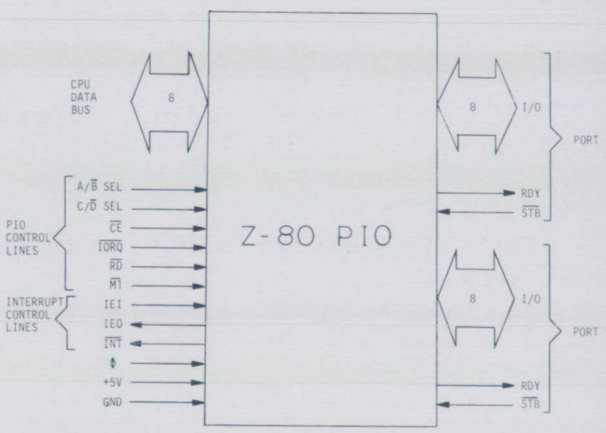


Figure 1. Functional Diagram of Z-80 PIO.

directions on port A. Mode 3, the control mode, is a hybrid of the input mode and the output mode; any line of the specified port can be designated as input or output. The control mode differs from the bidirectional mode in that once a line is designated as input or output, it stays in that condition and reprogramming is necessary to alter the direction of data transfer on that line. The handshake lines are not used in mode 3. A more detailed explanation of the Z-80 PIO operating modes can be found in references 2, 6 and 7 at the end of this article. This discussion is restricted to the control mode (mode 3). Other control registers internal to the PIO are used to store interrupt vector addresses, a distinguishing feature of the Z-80 PIO.

Let us now see how to construct and program a parallel interface for the ZX80/MicroAce computer, using this Z-80 PIO under non-interrupt, non-handshake control.

Construction of the Parallel Interface

Figure 2 shows a schematic diagram of the parallel interface. Port B is used for input to read the states of eight toggle switches (S1-S8) while port A is configured for output to drive eight light emitting diodes (D1-D8). Inverters are used to buffer the output port. The maximum output current capability of the PIO port data lines is about 1.5 milliamperes, not enough to drive an LED but enough to drive one TTL input or about four low power Schottky (LS) TTL inputs. The inverted system clock, $\overline{\Phi}$, is available at pin #6B on the back of the computer board. This signal is inverted again before being presented to the PIO. The handshake lines, $\overline{\text{STB}}$ and RDY, on each port are not connected. They are not needed because operating mode 3 will be selected. Since this application does not require interrupts, the IEI (Interrupt Enable In) line is tied high and the IEO (Interrupt Enable Out) is not connected.

Signals on the edge contacts of the computer board can be brought out through a cable using a modified 50-pin edge connector with 0.1 inch spacing between contacts (i.e., 3M part #3439-1000). The connector must be modified because it is closed ended and the computer requires an open ended version. The modification can be done with a sharp knife or a small saw.

The parallel interface circuit should be constructed on some sort of plug-in circuit board for easy inspection and modification. A high quality plug board such as Vector's 4677-2DP works well since it provides an etched power and ground bus. Etched pads for mounting dual-inline-plug (DIP) integrated circuits are also provided. All ICs should be socketed. Interconnections can be made by soldering small wires to the pads or by wire wrapping or a combination of both. I recommend the latter method: solder all power and ground lines to the appropriate pins on the wire wrap IC sockets and wire wrap control, data and address lines. Locate 0.1uF capacitors at each IC package, connected between +5V and GND, to decouple power supply spikes and suppress high frequency oscillations on the supply.

A suitable enclosure for the interface can be purchased from most electronic supply companies. It should be large enough to house a separate power supply which is required for operation of the interface. The circuit and power supply could also be mounted on a flat piece of material, such as aluminium or plexiglass, "open face" style.

There are two basic options for handling the power supply. If you are planning to add more circuitry to your system later, you should buy or build a relatively high current power supply. A schematic for a +5V, 3A power supply is shown in Figure 3. The regulated portion can be used to power the interface. The unregulated portion can be used to run the computer if you want to eliminate the standard calculator-type power supply. However, if you are not planning to add more active circuitry and you are satisfied with the calculator-type supply, you can get by without a separate supply. You will need though, a +5V voltage regulator to regulate the rough +9-11V going to the computer down to +5V for powering the interface which requires a total of about 100mA.

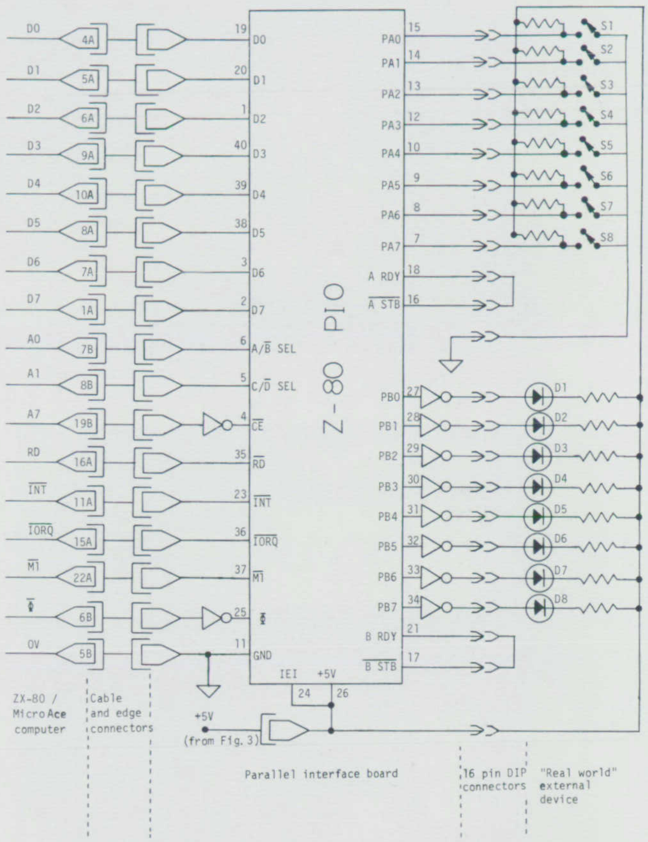


Figure 2. Schematic diagram of parallel interface circuit showing port A I/O lines being used as inputs and port B I/O lines as outputs. (Note: The software driver routine mentioned in the text assumes the opposite configuration; port A is output and port B is input.)

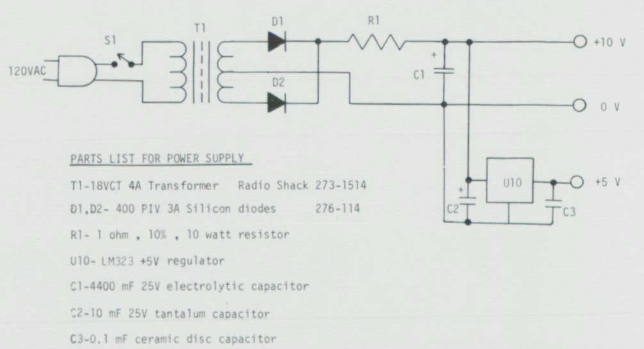


Figure 3. Schematic diagram of power supply used to operate the computer and the parallel interface circuit.

Figure 4. Author's 8K MicroAce system connected to a standard video monitor and standard size keyboard. The bread board in the foreground holds 16 LEDs which are used to monitor the outputs of both ports.

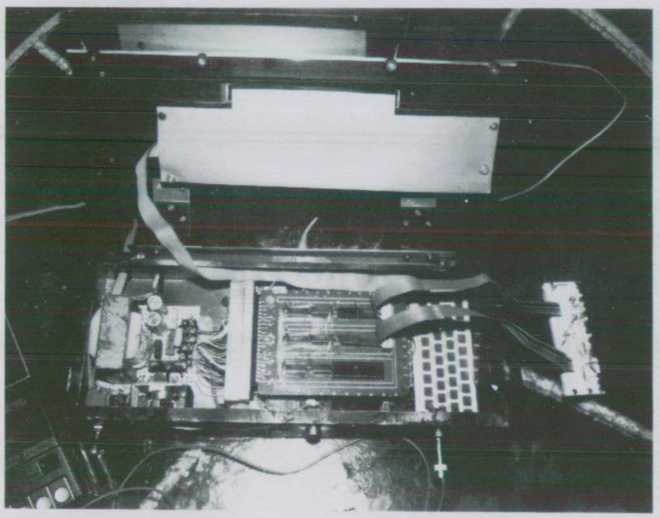


Figure 5. The inside of the author's system. The parallel interface and extra memory are located on the expansion board mounted above the computer.

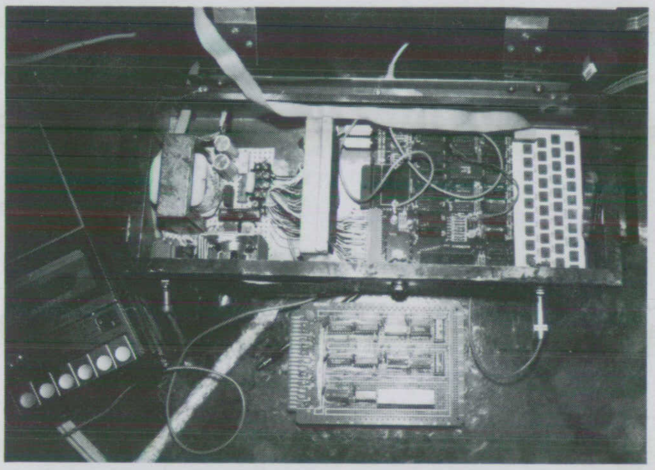


Figure 6. The inside with the expansion board removed, revealing the computer board.

Figures 4, 5 and 6 are photographs of the author's MicroAce system. The power supply provides unregulated +10V for the computer and 2K of on-board memory, and regulated +5V for the parallel interface plus an additional 6K of memory on the expansion board which is mounted just above the computer. The entire system is housed in a steel enclosure fitted with a hinged lid to which a standard size keyboard is mounted. The parallel port I/O (input/output) lines and handshake lines are brought out through two 16-pin DIP IC sockets. The cassette I/O connections are made available through two isolated phone jacks mounted on the front of the enclosure. Two RCA type phono jacks bring out the video signals: one for driving a standard video monitor and one for the RF modulator. Since the modulator is external to the computer, I use a TV as a video monitor for my other computer (Exidy Sorcerer).

Programming the PIO

Since the ZX80 version of Basic offers no direct means of communication with an I/O device, a driver subroutine coded in Z-80 machine language must be loaded into memory to operate the PIO. Data and control words can be passed from Basic to the driver routine through the POKE instruction. The routine is executed by calling it with a USR instruction. Some knowledge of the Z-80 CPU instruction set is helpful in understanding the driver routine.

Data is transferred from the CPU to the PIO by addressing one of its internal registers and writing to it by using one of the Z-80's OUT instructions. We need only be concerned with four of the PIO registers in this application: port A control, port B control, port A data, and port B data. Each register is accessed by a unique address. I/O instructions are always associated with one-byte addresses comprised of the lower 8 bits of the address bus. A minimum of 3 address bits is required to operate the PIO. Normally, address line A0 is connected to the port select line (B/$\overline{A}$ SEL) of the PIO and address line A1 is connected to the control/data select line (C/$\overline{D}$ SEL). The six remaining bits of the address byte are decoded to select one of a number of I/O devices. Since the PIO is the only I/O device in this system, decoding is not necessary. As shown in the schematic (Figure 2), address line A7 is inverted and connected to the chip enable line ($\overline{CE}$) of the PIO. Therefore, any address within the range 10000000B (B stands for binary) 11111111B will enable the PIO. The machine language driver routine (Figure 7) uses the "output immediate From Accumulator" instruction to transfer a byte of data to the PIO. This instruction is represented mnemonically

OUT(n),A

It transfers the contents of the Accumulator of the A register (one of the Z-80 CPU internal registers) to the I/O device addressed by n. The table in Figure 8 gives the addresses of the PIO internal registers and their significance when using the OUT immediate instruction in this configuration.

	Address		**Meaning**
Binary	*Hexadecimal**	*Decimal**	*Contents of Accumulator interpreted as...*
1XXXXX00	80	128	data — port A
1XXXXX01	81	129	data — port B
1XXXXX10	82	130	control — port A
1XXXXX11	83	131	control — port B
0XXXXXXX	00	0	PIO is not enabled, no change

X means "don't care"; this bit can be 1 or 0. * These values assume that X=0.

Figure 8.

Label Location	Machine (Dec)	code (Hex)	Mnemonic	Comment
0	0	00	NOP	Do nothing.
1	0	00	NOP	Do nothing.
2	62	3E	LD A, CFH	Load register A with operating
3	207	CF		mode control word.
4	211	D3	OUT (82H), A	Send control word to port A
5	130	82		control register.
6	62	3E	LD A, 00H	Load register A with data
7	0	00		direction word. All lines output.
8	211	D3	OUT (82H), A	Send data direction word to
9	130	82		port A control register.
10	62	3E	LD A, CFH	Load register A with operating
11	207	CF		mode control word.
12	211	D3	OUT (83H), A	Send control word to port B
13	131	83		control register.
14	62	3E	LD A, FFH	Load register A with data
15	255	FF		direction word. All lines input.
16	211	D3	OUT (83H), A	Send data direction word to
17	131	83		port B control register.
18	62	3E	LD A, 07H	Load interrupt control word.
19	07	07		
20	211	D3	OUT (82H), A	Send interrupt control word to
21	130	82		port A control register.
22	211	D3	OUT (83H), A	Send interrupt control word to
23	131	83		port B control register.
24	201	C9	RETN	Return to Basic program.
25	62	3E	LD A, 00H	Load register A with the
26	00	00		contents of this location.
27	211	D3	OUT (80H), A	Send contents of register A
28	128	80		to port A data register.
29	201	C9	RETN	Return to Basic program.
30	33	21	LD HL,, 0000H	Clear the HL register pair.
31	0	00		
32	0	00		
33	14	0E	LD C, 81H	Load register with port B
34	129	81		data register address.
35	237	ED	IN L, (C)	Read port B I/O lines. Load
36	104	68		data into register L.
37	201	C9	RETN	Return to Basic program.

Figure 7.

Before data can be sent through a port, certain control words must be loaded into the internal registers of the PIO. This process is called initialization, and the code that does this is called the initialization routine. Several things must be done in the initialization process: the operating mode must be set, the data direction must be established, and the interrupt servicing must be taken care of. In this example the selection of mode 3 simplifies matters since the handshake lines are not used. The operating mode is selected by writing a control word with the four least significant bits set high. The two most significant bits determine the opterating mode and the other two bits are not used as shown in Figure 9.

Operating Mode		**Control Word**		
		Binary	*Hexadecimal*	*Decimal*
Output	0	00XX1111	0	15
Input	1	01XX1111	4F	79
Bidirectional	2	10XX1111	8F	143
Control	3	11XX1111	CF	207

Figure 9.

When the control mode (mode 3) is selected for a particular port, the next control word sent to that port will define the direction of data transfer on each of the port's I/O lines. Each line corresponds to a bit position in the control word; the most significant bit of the control word corresponds to the most significant I/O line. A high condition (1) means input and a low condition (0) means output. For example, suppose the control word FOH (H stands for hexadecimal) is used to select data direction on the port B. Lines PBO through PB3 would be set up for output while lines PB4 through PB7 would be set up for input.

Interrupts are handled very conveniently in this application; they are disabled by simply writing 07H (00000111B) to the control registers in both ports.

The PIO machine language driver routine listed in Figure 7 may be located in the unused spare portion of memory. However, in order to save the driver on cassette tape it must be located in the variables area of memory which is located immediately following the user Basic program. (When a program is stored on tape only the program itself, system variables and program variables are saved; not all of memory.) The two memory locations 16392 and 16393 contain the low byte and high byte, respectively, of the starting address of the variables area. This address will be referred to by the symbol ORG which stand for origin. Since the value of ORG depends on the size of the Basic program, all addresses in the driver routine must be relative to ORG.

The driver consists of three machine language subroutines, each ending with a return from subroutine instruction. The first routine initializes the PIO, setting up port A for output and port B for input. (Note: This is opposite to what is shown in the schematic diagram of the parallel interface. This means that the switches should be connected to port B and the inverter-buffer inputs should be connected to port A.) The interrupts are disabled in the last portion of the initialization routine. Another routine sends a selected byte of the port A output routine. It is altered by the execution of a POKE instruction in the Basic program. The third routine, the port B input routine, reads the data present at the port B I/O lines and stores the information in the L register. The HL register pair is cleared, set to 0, at the beginning of the routine. Storing the data in the L register is convenient because, when a USR function is called, the value of the HL register pair is returned. For example, suppose that during execution of a Basic program the statement LET X=USR(Z) is encountered, where Z is equal to the starting address of a machine language routine that merely loads the value 31264 in the HL register pair. The variable X would then be equal to 31264 after the completion of the machine language routine. If the HL register pair was not altered during the routine, X would equal Z.

The Basic program that calls the driver routine must provide a means of entering the machine language code. Getting the code into the variables area is done by setting up an array, i.e., allocating a portion of memory (large enough to hold the driver) with a DIM statement. Getting the proper code into the array can be done in several ways. The simplest is to enter the elements as signed integers. Be aware that the integers are stored in two bytes of memory, with the less significant byte first. This makes it very difficult to decipher the machine code. A more elaborate method involves writing a Basic monitor which would include a hexadecimal-to-decimal routine and a decimal-to-hexadecimal routine for entering and displaying one-byte entries in hexadecimal notation. This would require perhaps more memory than a 1K machine could accommodate, but inspection and modification of the machine code would be much easier. The Basic program in Figure 10 employs the former method for entering the code.

```
  10 DIM M(20)
  20 LET V=16392
  30 LET ORG=PEEK(V)+PEEK(V+1)*256+2
  40 LET AO=ORG+25
  50 LET BI=ORG+30
  60 LET MLA=AO+1
  70 PRINT
  80 PRINT
  90 PRINT
 100 PRINT "MENU"
 120 PRINT "1) INPUT CODE"
 130 PRINT "2) REVIEW CODE"
 140 PRINT "3) PORT -A- OUT"
 150 PRINT "4) PORT -B- IN"
 160 INPUT A
 170 LET A=A*1000
 180 CLS
 190 GOSUB A
1000 FOR I=0 TO 20
1020 PRINT I
1040 INPUT M(I)
1060 NEXT I
1080 RETURN
2000 FOR I=1 TO 20
2020 PRINT I+ORG,PEEK(I+ORG),1+ORG+21,
     PEEK(I+ORG+21)
2040 NEXT I
2060 INPUT Z$
2080 RETURN
3000 PRINT "ENTER BYTE OUT"
3020 INPUT B
3040 POKE MLA,B
3060 LET X=USR(ORG)
3080 LET X=USR(AO)
3100 PRINT "ANOTHER BYTE ?"
3120 INPUT Z$
3140 CLS
3160 IF NOT Z$="" THEN RETURN
3180 GOTO 3000
4000 PRINT "HIT NEW LINE TO READ PORT B"
4020 INPUT Z$
4040 CLS
4060 IF NOT Z$="" THEN RETURN
4080 LET X=USR(ORG)
4100 LET X=USR(BI)
4120 PRINT "DATA AT PORT B...";X
4140 GOTO 4000
%
```

Figure 10.

This Basic program is menu-driven, giving the user the following options: entering the machine code, reviewing or listing the machine code, entering the byte to be sent out through the port A I/O lines, and reading the data present at the port B I/O lines. The variables used in the program are listed in Figure 11. The Sinclair version of integer Basic allows variables to be used as labels for GOTO and GOSUB statements. This feature is absent from many "expensive" versions.

Variable	**Meaning**
ORG	Beginning of driver routine.
AO	Beginning of port A output routine.
BI	Beginning of port B input routine.
MLA	Location of byte to be output through port A.
V,V+1	Points to the beginning of the variables area.

Figure 11.

Remember, when you get the driver routine loaded into memory, either by hand or by tape, *do not press RUN*. Instead, press GOTO followed by a number less than or equal to the lowest line number. GOTO 1 is safe.

Operation of the program is straightforward. The user is first shown a menu and is prompted to input a number between 1 and 4. If 1 is entered, the user is prompted to enter the signed integer elements which comprise the machine code. In this program the user should respond with the following integers. The screen is cleared after each entry.

Display	**Enter**	**Display**	**Enter**
0	0	10	-32045
1	-12482	11	-31789
2	-32045	12	16073
3	62	13	-11386
4	-32045	14	-13952
5	-12482	15	33
6	-31789	16	3584
7	-194	17	-4735
8	-31789	18	-13976
9	1854	19	0
		20	0

Figure 12.

Item 2 on the menu displays the contents of the array in which the driver routine is buried. Item 3 asks the user to input an integer, between 0 and 255, to be output through the port A I/O lines. For example, if the integer 255 is entered, all eight I/O lines will be set high; if 0 is entered, all lines will be set low. Selection of item 4 will read the port B data present at the port B I/O lines. If lines 0 and line are high and the others connected to ground, the decimal value "129" will be displayed. One way to exit the program is to break (hit the space key) while the screen is blank. Another way is to enter the letter "Z" when the computer is expecting an integer input, indicated by the appearance of the [LS] cursor in inverse video.

Applications

The number of possible applications for the ZX80/MicroAce with a parallel interface is limited only by the user's imagination. With 16 I/O lines, interfacing devices such as A/D (analog to digital) and D/A (digital to analog) converters to the computer is a possibility perhaps once thought unachievable by many ZX80/MicroAce owners. Control of high voltage-current devices is also possible with relays and relay driver circuits.

With an A/D converter one could realize an inexpensive data acquisition system for monitoring and recording various quantities in the laboratory, in industry, or in the home. For instance, a temperature to voltage or temperature to current converter could be connected to the A/D for recording temperatures over a period of time at specified intervals. The calibration could be done in software to reduce hardware costs. Of course, a simple voltmeter would also be a useful application.

A programmable voltage source or power supply could be constructed by connecting a D/A converter to the parallel interface. Complex waveforms can be generated by cycling through a table of data words to be output by the D/A thus providing the user with a programmable function generator. An A/D converter can even be realized by using a D/A and a voltage comparator in a configuration known as a successive approximation A/D converter.

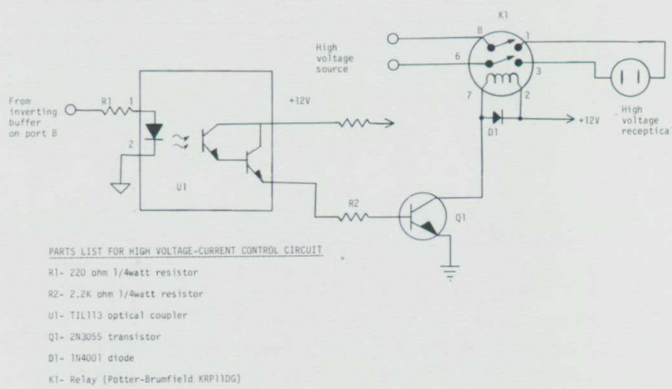


Figure 13. Schematic diagram of relay and computer controlled, optically isolated relay driver circuit.

Figure 13 shows a circuit that enables computer control of high voltage-current devices such as televisions, coffee makers, and lights, (NOTE: Use caution if you decide to build this circuit. All high voltage wires and connections should be isolated and insulated from the user and the computer circuits). In this circuit an output line from one of the inverting buffers is used to drive an optically isolated relay driver circuit. An optical coupler with a darlington transistor output is used to isolate the computer circuits from any high voltages which may appear. The darlington output provides higher current driving capability than a standard, single transistor optical coupler but at the sacrifice of speed which is of no consequence in this application. Any comparable relay with a 12 VDC coil will also work. Be sure to stay within the current ratings of the contacts, however. Relays with other DC (direct current) voltage ratings will also work with appropriate resistor-value substitutions in the circuit.

As I said, the possibilities are limitless. You may decide to just let the computer turn on LEDs in random sequence. At any rate, I hope you will experiment and share your discoveries with others via *SYNC*.

Selected References

1. Artwick, Bruce. *Microcomputer Interfacing*. Prentice-Hall, 1980.

2. Barden, William Jr. *The Z-80 Microcomputer Handbook*. Howard W. Sams and Co., 1979.

3. Engineering Staff on Analog Devices, Inc. *Analog-Digital Conversion Notes*. Analog Devices, 1976.

4. Nichols, Elizabeth; Nichols, Joseph; Rony, Peter. *Z80 Microprocessor Programing and Interfacing*. Books 1 and 2. Howard W. Sams and Co., 1979.

5. Salt, Alger. "Build Additional RAM." *Syntax* 2, no. 3 (March 1981).

6. *Z80 Assembly Language Programming Manual*. Zilog, 1977.

7. *Z-80-PIO Z80A-PIO Technical Manual*. Zilog, 1977.

See SYNC NOTES for a P.S. from the author.

Mini-Billboard

Dennis Duke

If you have spent much time looking at the schematic for your ZX80 or MicroAce, and if you have had the opportunity to compare it with schematics for other home computers, you probably have noticed that there are considerably fewer parts. This is due to the efficiency of circuit design in several areas. One of these areas involves the absence of separate character generator ROMs in the video circuitry.

The character generator is contained in the same ROM that holds the Basic interpreter. Sixty-four eight byte blocks are located in addresses 3584 to 4095 (0E00 to 0FFF hex). While the ZX80 is in the video display mode, the CPU is addressing these memory locations and loading the data into IC9 (U10 for MicroAce). This data transfer is parallel, or eight bits at a time. The data is then shifted serially, or one bit at a time. These bits go to the video modulator which causes either light or dark spots on screen, depending on whether the bit is a "one" or "zero." The IC21 (U14) keeps track of which byte is to be addressed by counting up to eight horizontal sync pulses to determine which of the eight horizontal lines for each character is being displayed.

You can examine each of the 64 characters in more detail by using the following program.

```
 10 INPUT A$
 20 LET A=(CODE(A$)*8)+3584
 30 FOR X=0 TO 7
 40 LET C=PEEK (A+X)
 50 FOR Y=0 TO 7
 60 LET E=2**(7-Y)
 70 IF C=E OR C)E THEN GO TO 100
 80 PRINT "  ";
 90 GO TO 120
100 LET C=C-E
110 PRINT CHR$(128);
120 NEXT Y
130 PRINT
140 NEXT X
PRESS RUN and NEWLINE. Then press
any key and NEWLINE.
```

Dennis Duke, 716 Torri Ct., Aledo, TX 76008.

Press RUN and NEWLINE. Then press any key and NEWLINE.

Line 20 converts the character A$ into the address of the first byte for that character in ROM. Line 40 sets C equal to the decimal value of that byte which is between 0 and 255 inclusive. In the first pass of the FOR-NEXT loop in lines 50 to 120, C is examined to determine if the most significant bit (MSB) of the data is a "one" or a "zero." If the MSB is a "zero," a space is printed. If the MSB is a "one," an inverse space is printed. In the next pass, the second most significant bit is examined and printed. The last pass will examine and print the least significant bit.

After eight bits have been printed, line 130 causes a new line so the next byte can be printed directly below the first. The FOR-NEXT loop in lines 30 to 140 causes eight bytes from sequential addresses to be printed.

The addition of another FOR-NEXT loop, an array, and some other modifications to this program allows us to print an eight character string on two rather large lines to create a "Mini-Billboard" on the TV screen.

Mini-Billboard

```
  5 DIM A(8)
 10 INPUT A$
 15 FOR I=1 TO 8
 20 LET A(I)=(CODE(A$)*8)+3584
 21 LET A$=TL$(A$)
 23 NEXT I
 25 LET F=1
 27 LET L=4
 30 FOR X=0 TO 7
 35 FOR I=F TO L
 40 LET C=PEEK (A(I)+X)
 50 FOR Y=0 TO 7
 60 LET E=2**(7-Y)
 70 IF C=E OR C)E THEN GO TO 100
 80 PRINT " ";
 90 GO TO 120
100 LET C=C-E
110 PRINT CHR$(128);
120 NEXT Y
130 NEXT I
140 NEXT X
150 LET F=F+4
160 LET L=L+4
170 IF L=8 THEN GO TO 30
```

Press RUN and NEWLINE. Then enter READ SYNC (or any two four letter words) and NEWLINE.

You probably noticed we no longer need a PRINT statement in line 130 since four groups of eight characters are now printed in a line which will cause an automatic new line by coming to the end of a 32 character line. If you want to use a different graphic, change the number in line 110. Try also 7, 136, 8, and 223.

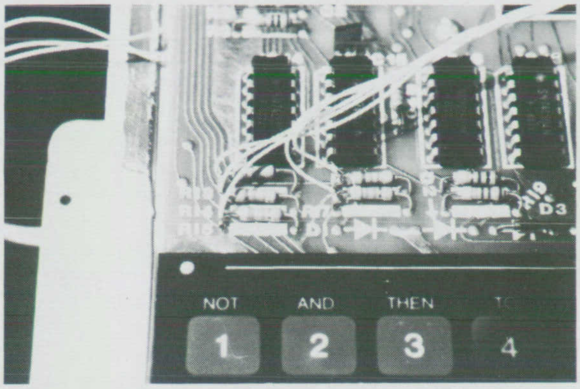
So now you have a program which will print two large, four letter words on your TV screen. This may lead to some interesting suggestions from your friends, but have fun with it anyway.

Hardware Review

Keyboard Beeper

Joe Utasi

Anyone with a Sinclair or MicroAce has experienced the hassle of having to check the TV screen after every entry to see if it got into the machine. Of course, there are those people with good peripheral vision who can manage this feat without bobbing their heads, but not me. So when I saw an ad for a "keyboard beeper," I realized this most certainly would be a big help in entering programs on the membrane keyboard and sent for one.



The beeper comes assembled and is extremely simple. It consists of two integrated circuits, two resistors, and a capacitor mounted to a P.C. board barely larger than the components. The power and ground wires are connected to the ZX80 board just below the modulator on some wide power traces. Five wires (which were twisted into a bundle) go to the keyboard side of the five pullup resistors at the extreme lower left side of the board. The order of sequence does not matter, as long as you connect to the side of the resistors that goes to the keyboard and *not* power. It is easy to see which side goes to the keyboard by just following the traces.

The last step is to install the small round piezo-electric transducer which produces the sound. The directions provided with the beeper suggest soldering one edge to the top of the modulator (the left side), so that is where I put it. The one remaining wire from the beeper board is soldered to the white portion of the transducer. I used a piece of carpet tape (not included in the package) to mount the beeper to the inside of the case top on the front surface of the "blister."

The beeper worked perfectly the first time. Slight changes in the tone of the beep for different keys can be detected. This might be an asset if you have a good ear.

The real advantage comes when entering SHIFTed commands. Programming seems to go faster with less aggravation now that I know I am making good "contact" with the keyboard. I would certainly recommend the beeper as a definite improvement to the ZX80.

Keyboard Beeper, $12.
Burnett Electronics
908 Morris St.
Cincinnati, OH 45206

Joe Utasi, 2028 Knightsbridge Dr., Cincinnati, OH 45244.

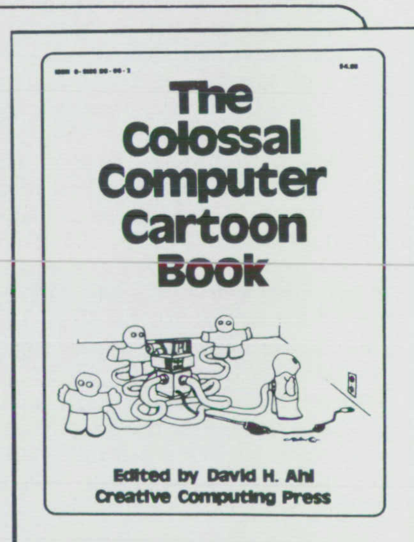
Do Computer Enthusiasts Have More Fun?

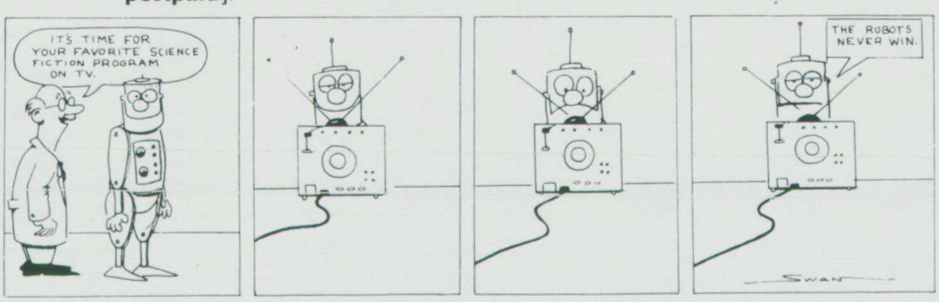
The Colossal Computer Cartoon Book

The best collection of computer cartoons ever is now in its second printing, and sports a bright new cover. The fifteen chapters contain hundreds of cartoons about robots, computer dating, computers in the office, home, and lab, and much more. 36 cartoonists share their views of man's ultimate machine.

Keep this book with your reference works. When needed, the right cartoon can say it all for you. When you need a break from debugging a good laugh can give you a welcome lift. Recommended for hours of fun and comic insight.

Edited by David Ahl, mastermind behind the April Fool's issue of Dr. Kilobyte's Creative Popular Personal Recreational Micro Computer Data Interface World Journal, this cartoon book contains much of that same incurable zaniness. [Want this issue? It's April 1980 and only $2.50 postpaid].





A large 8½ x 11" softbound collection of 120 pages, it still sells for only $4.95. (6G).

8K Basic ROM

David Lubar

While the 4K Integer Basic in the Sinclair ZX80 is adequate for many applications, most programmers will eventually feel a hunger for more power. True, advanced functions can be simulated by way of subroutines, but such measures eat memory at an alarming rate. Enter the 8K Basic ROM. The chip costs a mere $39.95, which is an extremely low price for any ROM. Some versions of Basic are sold for over $200 on disk. Sinclair gets four stars for not robbing its customers.

Plugging In

Installing the ROM chip requires opening the Sinclair. Most owners have probably already done this out of curiosity and learned that nothing disasterous follows. One really has to go out of his way to hurt the little critter. The only problem is dealing with the plastic pins which hold the case together. Once the case is open, the old ROM has to be removed. This requires some patience. If a chip is pulled with unequal pressure, the pins can be bent. It's best to keep the old ROM intact, for reasons that will be covered later. The new ROM is installed by lining up the pins and exerting *gentle* pressure. Next, a new keyboard overlay is put in place. This overlay contains letters, numbers, keywords, graphics symbols and functions, with color coding to aid the confused. Once the ROM has been tested by powering up the computer, the case can be replaced.

Features

With the ROM installed, the Sinclair has floating point capability. It can handle decimals with nine-place accuracy. Other added functions include string and numeric arrays of any dimension, trig functions, and extended string functions. The PLOT and TAB commands allow formatting of text and graphics. Unfortunately, the proposed DRAW command, which would have drawn a line between any two sets of coordinates, was not included in the final version of the Basic.

As before, keywords are obtained with a single stroke. By hitting the FUNCTION key, the user can also obtain functions with one keystroke. Don't get excited about the commands FAST and SLOW. The Sinclair already operates in the FAST mode. The SLOW mode (to eliminate flicker of the display) only works on the ZX 81, which is not yet available in the U.S. There is a SCROLL command, which moves the screen display up one line. The computer will still crash if you attempt to write beyond the screen.

Several commands have been provided for use with the printer Sinclair plans to introduce. The user will be able to send listings to this printer and to print the contents of the screen. For interactive programs, there is an INKEY$ command. This reads the keyboard without requiring NEWLINE. The pause command sends the contents of the display list to the screen and waits a specified amount of time. This allows for limited animation, but still produces a flicker. All in all, the 8K Basic greatly expands the potential of the Sinclair.

Compatability

The 8K ROM contains an improved set of tape routines. While this means that loading and saving should be less hassle, it also means that you can't load old-ROM tapes into a new-ROM machine. And even if you could load such programs, they wouldn't run. This means most users will be doing a lot of translating. Two major differences must be kept in mind. First, many programs took advantage of the Integer Basic, ignoring the remainder after division. To simulate this in the new Basic, use the INT function. Secondly, where the Integer Basic ROM used minus one for true when evaluating logical operations, the 8K ROM uses positive one. Any calculations based on logical operators will require a sign change during translation.

Ideally, it would be nice to be able to switch from one ROM to the other. Someone is bound to produce such a switch in the near future and many enterprising hobbyists are likely to design their own. While such a switch would clobber anything in memory, it would allow loading of either flavor of tape without pulling and replacing chips. For this reason, it is advisable to hold onto the old ROM.

The most noticable difference between the ROMs occurs when you try entering a 1K program. The new ROM uses about 100 bytes more of RAM than the old ROM. Most programs that fit into 1K before won't fit now. To get any value out of the new Basic, a user should have at least 2K, preferably 16K.

So, if you are feeling limited by 4K Basic, and either plan to expand memory, or already have, then the 8K Basic ROM is an excellent way to extend the capabilities of your Sinclair. The 8K Basic ROM is available for $39.95 plus $4 shipping from Sinclair Research Ltd., 1 Sinclair Plaza, Nashua, NH 03061.

try this

This column will feature short programs to show off your ZX80, impress your family and friends, and tickle your imagination when *SYNC* arrives at your place. We invite your contributions. Address them to *SYNC*, 39 E. Hanover Ave., Morris Plains, NJ 07950.

```
 10 LET M=16567
 20 FOR A=386 TO 419
 30 POKE M+A-386,PEEK(A)
 40 NEXT A
 50 POKE M+A-386,201
 60 FOR A=0 TO 32767
 70 PRINT A
 80 LET B=USR(M)
 90 CLS
100 NEXT A
```

Notes:
10 A few bytes after DF-END
20 Section in Basic to turn on screen
50 Return at end
Enter RUN and NEWLINE. You will have to adjust the screen to get as good a picture as possible, but it still will not be perfect.

Our thanks to:
David Goodrich
124 NE Spruce
Bartlesville, OK 74003

And the Walls Came Tumbling Down

David and James Grosjean

After the successful introduction of *Super ZX80 Invasion*, [see *SYNC* 3:5] Softsync has come out with *Double Breakout*, its second active display game. *Double Breakout* is just as much fun as *Super ZX80 Invasion*, and even more challenging. This, too, fits into 1K of memory.

After loading the game from the cassette, the words "100 REM" appear at the top of your screen. Enter "GO TO 1" and then select your level of play. There are seven skill levels where 7 is slow enough for beginners, 4 is medium, and 1 is extremely fast for the expert. Softsync's brochure claims that you do not have a chance at level 1, but we have found that after extensive play you do have a good chance.

A game field 31 spaces wide and 18 spaces high appears on the screen. Within the area are two walls of blocks running vertically, each five rows thick. One is in the middle of the screen, and the other is off to the right. The paddle appears in the upper left hand corner of the screen and can be moved up and down along the left side by using the arrow keys (5 and 8). The makers recommend that your computer be turned sideways so the keys will face up and down according to the movements of the paddle, but we suggest that you turn your television sideways if possible. The ball, represented graphically by the letter "0", bounces between your paddle and the blocks, each time chipping a block off the wall. Once you break through the first wall there is another wall which you must also knock out.

David and James Grosjean, 50 Kings Rd., Chatham, NJ 07928.

You have nine balls with which to knock out the blocks. The number of balls remaining is displayed in the left hand corner of the screen, just outside of the playing area. Each time you miss the ball, the number diminishes by one, and the next ball is served immediately. If you lose all the balls, a new game is started, and, if you successfully clear out all the blocks, the ball continues to bounce around.

SYNC

SOFTWARE PROFILE

Name: Double Breakout

Type: Arcade Game

System: Sinclair ZX80; MicroAce. 4K ROM

Format: Cassette

Language: Basic

Summary: Even more challenging than *Super ZX80 Invasion*

Price: $14.95 plus $1.50 shipping

Manufacturer:
SOFTSYNC, INC.
P.O. Box 480
Murray Hill Station
New York, NY 10156

You cannot stop the game to change skill levels during play. The BREAK key does not function. You must unplug the machine and reload the game.

By deleting line 100, the portion of the program written in Basic is revealed. This is the part which asks for the ball speed and then calls a machine code subroutine which actually plays the game. Line 450 of the program makes sure you do not enter a speed slower than 7 or faster than 1. If you delete this line, you can enter a speed slower than 7. The game will run the same as before even with line 100 missing.

For those of you who play the original arcade *Breakout* games by Atari, here are a few comparisons: The name *Double Breakout* does not mean two balls and two paddles, like Atari's, but two walls of blocks. This could be confusing. In Atari's arcade *Double Breakout*, the ball increases speed as it hits more blocks, but in this *Double Breakout* the speed you choose at the beginning of the game remains the same. Softsync's *Double Breakout* gives nine balls with automatic serving, while the arcade game gives only three balls with manual serving. *Double Breakout* serves a new ball as soon as one is lost, so on level 1, if a ball is lost, the next one will be served so quickly, that you might not be able to get to it in time return it.

One shortcoming of the game is that there is is no scoring, and another is that there is no extended play such as extra balls or walls.

Double Breakout is another breakthrough in creating active display games for the ZX80. We had great fun playing *Double Breakout* and are amazed at how much they fit into 1K.

resources

Software

- ZX80/MicroAce software on cassette: *Dragon Castle Adventure, Betting System for Horse Players, Robot Composer,* and *ESP Guessing*; all 4 for $10.
  Cecil Bridges
  1248 N. Denver
  Tulsa, OK 74106

- Three cassette tapes: (1) *Slot Machine, Robot Fight, Corporation, Tank Battle;* (2) *Lucky Lindy, Crop Duster, Nuke Em, Carrier Landing;* (3) The Pharaohs Treasure; $10 each.
  Tensor Technology Inc.
  P.O. Box 17868
  Irvine, CA 92713

- *Smart Reversi* [Othello]. Play the classic game against your ZX80. Uses a very strong move algorithm extracted from a much larger program; game board display. (Othello [R] is a trademark of CBS Toys, Inc.) $6. £3.50 (U.K.)
  C. W. Percival
  193 Peaceable St.
  Ridgefield, CT 06877

- The ZX80 Companion is now available with a 20 pp. supplement for the ZX81. The supplement is available separately for £1.50.
  Linsac
  68 Barker Road
  Linthorpe
  Middlesbrough, Cleveland, TS5 5ES
  England

- ZX80 Multiple Line Statements
  Easy, Useful Programming Trick.
  Saves memory, runs faster. Details £1 inc. postage (U.K.) or $2.50 inc. airmail (USA).
  Tim Humphries
  16 Coniston Road
  Sutton Coldfield
  West Midlands
  England

- *ZX80 Graphics*; 48 pp. containing programming techniques and 6 original programs; $8 incl. postage.
  SUMWARE
  P.O. Box 30
  Shawville, PA 16873

- ZX80/1 *Record*; a tape record system to save, load, or enter new 96 byte records; ideal for addresses; for all 1K machines (4K/8K ROM); £3; $9. *Directory*; a simple program to read tapes and display program names; (8K ROM); £2; $6.
  Logan Software
  24 Nurses Lane
  Skellingthorpe
  Lincoln LN6 OTT
  UK

- Music cassette: Side A: Player ZX80; Side B: space MUSE-AK, a random sound program. Prepaid orders; $6.95 postpaid ($10 outside the U.S.). Other programs available.
  William Don Maples
  688 Moore St.
  Lakewood, CO 80215

- 5 cassettes: (1) Games; (2) Junior Education; (3) Business and Household; (4) Games; (5) Junior Education; £3.95. Designed for the ZX81, but many will run on the ZX80 with 8K ROM; some need the 16K RAM pack. Cheque/PO; Access/Barclaycard.
  Sinclair Research
  FREEPOST 7
  Cambridge, CB2 1YY
  UK

- *Microcomputer Index*; subject indexing of articles in 20 microcomputer periodicals.
  Microcomputer Information Services
  2464 El Camino Real
  Box 247
  Santa Clara, CA 95051

- Filing program "Multifile"; £17.50. Machine code assembler "ZXAS" for ZX80 or ZX81 (specify); £3.95.
  Bug-Byte
  251 Henley Road
  Coventry CV2 1BX
  England

- Compute and display program (1K & 2K) with instruction booklet, coding sheets, and coding charts for ZX80 (4K ROM); £4.95.
  JRS Software
  19 Wayside Avenue
  Worthing
  West Sussex, BN13 3JV
  England

- Wide range of games for ZX80/1 (4K & 8K ROM).
  Premier Publications
  12 Kingscote Road
  Addiscombe, Croydon,
  Surrey
  England

- 2 versions of *Defender* with built in software to drive their soundboard (£25); £4.50 small screen; £5.50 large screen.
  Quicksilva
  9S Upper Brownhill Road
  Maybush, Southampton,
  Hants
  England

Hardware

- Re-Zolv Resist. Used with positive or negative transparencies; circuit patterns can be drawn also; develops with water. For the hobbyist and the professional engineer. Starter kit: COD $13.40; $12 for prepaid. Phone (217) 352-9336.
  Coval Industries, Inc.
  2706 W. Kirby Ave.
  Champaign, IL 61820

- MicroAce upgrade products:
  8K ROM; $35.
  Video upgrade board for flicker free display (8K ROM required); $29.50.
  MicroAce 2K Computer Kit; $149.
  Planned for the fall: 16K RAM, $150; 4K RAM, $110.
  MicroAce
  1348 E. Edinger
  Santa Ana, CA 92705

Users Groups

- ZX80 Southeast Region Club
  869 Levitt Parkway
  Rockledge, FL 32955
  Newsletter planned beginning in August. Pres. Ralph Coletti. Inquiries from interested parties welcome.

Creative Computing-- Albert Einstein in black on a red denim-look shirt with red neckband and cuffs.

Creative's own outrageous **Bionic Toad** in dark blue on a light blue shirt for kids and adults.

Plotter display of **PI to 625 Places** in dark brown on a tan shirt.

I'd rather be playing spacewar-- black with white spaceships and lettering.

Give your tie a rest!

All T-shirts are available in adult sizes S,M,L,XL. Bionic Toad, Program Bug and Spacewar also available in children's sizes S(6-8), M(10-12) and L(14-16). Made in USA. $6.00 each plus 75¢ shipping.

Specify design and size and send payment to Creative Computing, 39 E. Hanover Ave., Morris Plains, NJ 07950. Orders for two or more shirts may be charged to Visa, Master-Card or American Express. Save time and call toll-free **800-631-8112** (in NJ 201-540-0445).

Crash Cursor and Sync from the comic strip in SYNC magazine emblazoned in white on this black shirt.

Computer Bum-- black design by cartoonist Monte Wolverton on gray denim-look shirt with black neckband and cuffs.

The **Program Bug** that terrorized Cybernia in Katie and the Computer is back on this beige t-shirt with purple design. You can share the little monster with your favorite kid.

Roll down the block with this little black **Robot Rabbit** (on a bright orange t-shirt) on your back and you can intimidate every carrot, radish or cuke in your way.

...A SCIENCE-FICTION CARTOON COMEDY SERIES IN EVERY ISSUE OF...

SYNC

PLUS: Articles, games, applications, reviews and MORE!

Brought to you by the people at

creative computing

SYNC is the dynamite bi-monthly magazine for users of the Sinclair ZX80. The main focus is on applications, programming techniques, hints and tips for getting the most out of the ZX80. SYNC also reviews new peripherals, software and books for the ZX80. Subscriptions to SYNC cost just $10 for six bi-monthly issues (£10 in the U.K.). Send to SYNC, 39 E. Hanover Avenue, Morris Plains, NJ 07950, USA.